

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

AMERICAN AIRLINES, INC. AND SOUTHWEST AIRLINES CO.,

Petitioner,

v.

THE REGENTS OF THE UNIVERSITY OF CALIFORNIA,

Patent Owner.

IPR2025-01511

PATENT NO. 7,712,080

Systems and Methods for Parallel Distributed Programming

DECLARATION OF MICHAEL IAN SHAMOS, PH.D., J.D.

TABLE OF CONTENTS

I. INTRODUCTION AND QUALIFICATIONS	1
II. MATERIALS CONSIDERED	5
III. SUMMARY OF OPINIONS	7
IV. LEGAL PRINCIPLES	7
V. OVERVIEW OF THE '080 PATENT.....	8
VI. LEVEL OF ORDINARY SKILL IN THE ART	11
VII. CLAIM CONSTRUCTION.....	13
VIII. BACKGROUND OF THE PRIOR ART	13
A. The MESSENGERS Project	14
B. Fukuda '96.....	15
C. Fukuda '96b.....	15
D. Fukuda '97a.....	16
E. Fukuda '97b.....	17
F. Fukuda '98 (EX1008).....	18
G. Wicke '98	22
H. Fukuda '01 (EX1009).....	23
I. Pan '01 (EX1005).....	26
J. Pan '02.....	26
K. Art Raised During Prosecution	28
L. Conclusions Regarding the Prior Art	30
IX. THE CHALLENGED PATENT.....	31
X. Ground 1: Claims 1-18 ARE ANTICIPATED BY FUKUDA '98 (EX1008) .37	
A. Claim 1	37
1. [1pre] A method of developing a distributed parallel computing program, comprising steps of:	38
2. [1a] establishing at least one distributed shared variable, wherein the at least one distributed shared variable is a single variable that includes several variables that may be physically distributed across multiple memories;	38

3. [1b] developing at least one distributed sequential computing program to access the at least one distributed shared variable;	41
4. [1c] transforming the at least one distributed sequential computing program into at least one distributed parallel computing program by spawning at least one child distributed sequential computing program from the at least one distributed sequential computing program when at least one intermediate condition occurs within the at least one distributed sequential computing program	43
5. [1d] wherein the at least one distributed parallel computing program concurrently uses the at least one distributed sequential computing program and the at least one spawned child distributed sequential computing program to perform parallel processing and/or operations, ...	48
6. (e) generating at least one output combining all of the subtask outputs and reflecting the processing of all of said subtasks.	49
B. Claim 2: The method of claim 1, wherein each of the at least one distributed sequential computing program includes at least one self-migrating thread, capable of migrating among at least one processor and capable of accessing the at least one distributed shared variable.	52
C. Claim 3: The method of claim 1, wherein the at least one distributed shared variable is configured to be located among a plurality of memories	54
D. Claim 4: The method of claim 1, wherein the distributed parallel computing program is configured to operate across multiple processors.	55
E. Claim 5: The method of claim 1, wherein the distributed parallel computing program is configured to operate using multiple threads.	56
F. Claim 6: The method of claim 1, wherein the distributed parallel computing program is configured to operate across multiple nodes.	57
G. Claim 7: The method of claim 1, further comprising the step of configuring the at least one distributed sequential computing program to include at least one mobile agent.	58
H. Claim 8: The method of claim 7, wherein the at least one mobile agent is implemented using self-migrating threads	59
I. Claim 9	60
1. [1pre] A distributed parallel computing system, having at least one memory area and at least one processor, comprising:	60
2. [9a] at least one distributed shared variable capable of loading into the at least one memory area, wherein the at least one distributed shared variable is a	

single variable that includes several variables that may be physically loaded into the at least one memory area;	61
3. [9b] at least one distributed sequential computing program, configured to operate in the at least one processor, configured to access the at least one distributed shared variable, ...	61
4. [9c] configured to transform into at least one distributed parallel computing program by spawning at least one child distributed sequential computing system program when at least one intermediate condition occurs within the at least one distributed sequential program, ...	61
5. [9d] wherein the at least one distributed parallel computing program concurrently uses the at least one distributed sequential computing program and the at least one spawned child distributed sequential computing program to perform parallel processing and/or operations,.....	62
6. [9e] wherein the at least one distributed parallel computing program concurrently uses the at least one distributed sequential computing program and the at least one spawned child distributed sequential computing program to perform parallel processing and/or operations,.....	62
J. Claim 10: The system of claim 9, wherein the at least one distributed sequential computing program comprises at least one mobile agent.	62
K. Claim 11: The system of claim 10, wherein the at least one mobile agent is implemented using self-migrating threads.	63
L. Claim 12: The system of claim 10, wherein the at least one mobile agent is an explicit-navigation mobile agent.	63
M. Claim 13: The system of claim 10, wherein the at least one mobile agent is an implicit-navigation mobile agent.	64
N. Claim 14: The system of claim 10, wherein the at least one mobile agent is configured to move from one processor to another processor.	65
O. Claim 15: The system of claim 10, wherein the at least one mobile agent operates within a distributed shared memory system.....	66
P. Claim 16: The system of claim 9, wherein the at least one processor is located across a physical network.	66
Q. Claim 17: The method of claim 1, wherein the distributed parallel computing program is configured to operate across a distributed shared memory system.	67
R. Claim 18: The system of claim 10, wherein the at least one mobile agent is configured to move from one processor to another processor.	67

XI. CONCLUDING STATEMENT 67

XII.JURAT 67

I, Michael I. Shamos, declare as follows.

I. INTRODUCTION AND QUALIFICATIONS

1. I have been retained by McKool Smith P.C. on behalf of Petitioner American Airlines, Inc. (“American”) and by Munck Wilson Mandala LLP on behalf of Petitioner Southwest Airlines Co. (“Southwest”) (collectively, “Petitioners”) to provide my opinions concerning the validity of claims 1-18 (the “Challenged Claims”), constituting all claims of U.S. Patent No. 7,712,080 (“’080 Patent,” or the “Patent,” EX1001) in support of Petitioner’s “Petition for *Inter Partes* Review of U.S. Patent 7,712,080 B2” (the “Petition”).

2. I have been retained as an independent expert consultant in this proceeding before the United States Patent and Trademark Office. Although I am being compensated at my rate of \$625.00 per hour for the time I spend on this matter, no part of my compensation depends on the outcome of this proceeding, and I have no other interest in this proceeding, any of the parties, or the subject patent.

3. A detailed description of my professional qualifications, including a listing of my specialties/expertise and professional activities and publications, is contained in my curriculum vitae, a copy of which is provided as EX1014. Below is a short summary of my professional qualifications.

4. I received an A.B. (1968) from Princeton University in Physics; a M.A. (1970) from Vassar College in Physics; a M.S. (1972) from American University in

Technology of Management, a field that covers quantitative tools used in managing organizations, such as statistics, operations research and cost-benefit analysis; a M.S. (1973), and M.Phil. (1974) and a Ph.D. (1974) from Yale University in Computer Science; and a J.D. (1981) from Duquesne University.

5. I currently hold the title of Distinguished Career Professor in the School of Computer Science at Carnegie Mellon University in Pittsburgh, Pennsylvania. I am a member of two departments in that School, the Software and Societal Systems Department and the Language Technologies Institute. I was a founder and Co-Director of the Institute for eCommerce at Carnegie Mellon from 1998-2004 and from 2004-2018 was Director of the eBusiness Technology graduate program in the Carnegie Mellon University School of Computer Science. Since 2018, I have been Director of the M.S. in Artificial Intelligence and Innovation degree program at Carnegie Mellon.

6. I have taught graduate courses at Carnegie Mellon in Electronic Commerce, including eCommerce Technology, Electronic Payment Systems, Electronic Voting, Internet of Things, Electronic Payment Systems and eCommerce Law and Regulation, as well as Analysis of Algorithms. Since 2007 I have taught an annual course in Law of Computer Technology. I currently also teach Artificial Intelligence and Future Markets.

7. I am the author and lecturer in a 24-hour video course on Internet protocols and have taught computer networking, wireless communication, and Internet architecture since 1999.

8. From 2001-2021, I was a Visiting Professor at the University of Hong Kong, where I taught an annual course in Electronic Payment Systems. This is one of only a handful of graduate courses taught on this subject in the world.

9. I was the Director of Carnegie Mellon's graduate degree program in eBusiness Technology from 1999-2018 and am now a faculty member in the Privacy Engineering degree program at Carnegie Mellon. My course on Law of Computer Technology is required for all students in that program, the M.S. in Artificial Intelligence and Innovation program and in the Societal Computing Ph.D. program.

10. From 1979-1987, I was the founder and president of two computer software development companies in Pittsburgh, Pennsylvania, Unilogic, Ltd. and Lexeme Corporation.

11. I am an attorney admitted to practice in Pennsylvania and have been admitted to the Bar of the U.S. Patent and Trademark Office since 1981. I have been asked to render opinions in this Declaration as a technical expert. I have not been asked to offer any opinions on patent law in this proceeding.

12. I am a named co-inventor on the following six issued patents relating to electronic commerce: U.S. Patent Nos. 7,330,839, 7,421,278, 7,747,465, 8,195,197, 8,280,773, and 9,456,299.

13. I have previously served as an expert in over 400 cases concerning computer technology.

14. I am aware of contemporaneous litigation involving Petitioners and the Patent in cases styled *Intellectual Ventures I LLC et al. v. American Airlines, Inc.*, 4:24-cv-980-ALM (E.D. Texas) and *Intellectual Ventures I LLC et al. v. Southwest Airlines Co.*, 7:24-cv-277 (W.D. Texas) (the “Litigations”).

15. This IPR petition involves the ’080 Patent, which was filed on May 21, 2004 and entitled “Systems and Methods for Parallel Distributed Programming.” The Patent claims priority to U.S. Provisional Patent Application 60/472,612 (the “’612 Provisional”), filed May 21, 2003. The term of the Patent was extended by 1013 days, so it will expire on February 28, 2027.

16. I have been asked to assume, for purposes of this IPR proceeding only, that the claims of the Patent are all entitled to a Priority Date of May 21, 2003.

17. I have no financial interest in Patent Owner or the ’080 Patent. It is conceivable that I may own mutual funds whose portfolios include stock in one or both Petitioners. If this is the case, the value of such holding would not constitute a material part of my net worth.

18. The statements made and opinions provided in this Declaration are based on my own personal knowledge and, if called as a witness, I could and would testify in a manner consistent with this Declaration.

19. In this Declaration, all emphasis has been added unless otherwise noted.

II. MATERIALS CONSIDERED

20. In forming my opinions, I have reviewed and am familiar with the '080 Patent and its prosecution history, the materials cited herein, and the following exhibits to the Petition.

Exhibit No.	Description
1001	U.S. Patent No. 7,712,080 B2 (the '080 Patent) to Lei Pan, Lubomir R. Bic, and Michael B. Dillencourt
1002	Prosecution history of the '080 Patent
1003	This declaration
1004	CV of Michael Shamos
1005	Lei Pan, Lubomir F. Bic, Michael B. Dillencourt, <i>Distributed Sequential Computing Using Mobile Code: Moving Computation to Data</i> , International Conference on Parallel Processing, pp. 77-84, September 3-7, 2001, available at https://ieeexplore.ieee.org/document/952049 ("Pan").
1006	Naoya Suzuki, Munehiro Fukuda, Lubomir F. Bic, <i>Self-Migrating Threads for Multi-Agent Applications</i> , ICWC 99; IEEE Computer Society International Workshop on Cluster Computing, December 2-3, 1999, available at https://ieeexplore.ieee.org/document/810828 ("Suzuki").

Exhibit No.	Description
1007	Munehiro Fukuda, Lubomir F. Bic, Michael B. Dillencourt, Fehmina Merchant, <i>Message versus Messengers in Distributed Programming</i> , Proceedings of 17th International Conference on Distributed Computing Systems, May 27-30, 1997, available at https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=1c575e92687dbc06acec75ae66a7a2f9174cb31c (“Fukuda ’97”)
1008	Munehiro Fukuda, Lubomir B. Bic, Michael B. Dillencourt, Fehmina Merchant, <i>Distributed Coordination with Messengers</i> , Science of Computer Programming 31 (July 1998) 291-311, available at https://www.sciencedirect.com/science/article/pii/S0167642397000245 (“Fukuda ’98”).
1009	Munehiro Fukuda, Lubomir F. Bic, Michael B. Dillencourt, Fehmina Merchant, <i>Messengers: Distributed Programming Using Mobile Agents</i> , Transactions of the Society for Design and Process Science, Vol. 5, No. 4, pp. 95-112, December 2001, available at https://scispace.com/papers/messengers-distributed-programming-using-mobile-agents-50ktvx5clr (“Fukuda ’01”).
1010	Amended Complaint, <i>Intellectual Ventures I LLC v. American Airlines, Inc.</i> , No. 4:24-cv-980-ALM (E.D. Tex. Nov. 2, 2024) (Dkt. No. 84)
1011	Amended Complaint, <i>Intellectual Ventures I LLC v. Southwest Airlines Co.</i> , No. 7:24-cv-277-ALM (W.D. Tex. Nov. 2, 2024) (Dkt. No. 56).
1012	Infringement Contentions re the ’080 Patent in <i>Intellectual Ventures I LLC v. American Airlines, Inc.</i> , No. 4:24-cv-980-ALM (E.D. Tex. Nov. 2, 2024).
1013	Infringement Contentions re the ’080 Patent in <i>Intellectual Ventures I LLC v. Southwest Airlines Co.</i> , No. 7:24-cv-277-ALM (W.D. Tex. Nov. 2, 2024).

21. I have also relied on my education, skill, training, and experience in the relevant fields of technology. I have further considered the viewpoint of a person of

ordinary skill in the art (“POSITA”) in the subject matter of the ’582 Patent as of March 13, 2002.

22. I reserve the right to supplement my opinions as expressed in this Declaration to address any new information obtained in the course of this proceeding, or based on any new positions taken by Patent Owner.

III. SUMMARY OF OPINIONS

23. It is my understanding that the following table summarizes the sole ground raised in the Petition:

Grd.	Sec.	References	Challenged Claims
1	102	Fukuda ’98	1-14

24. After a review of the ’080 Patent and the prior art asserted by Petitioners, it is my opinion that the Challenged Claims are invalid as anticipated by Fukuda ’98. My opinions, and the bases therefor, are detailed throughout this Declaration.

IV. LEGAL PRINCIPLES

25. Counsel for Patent Owner has informed me of the legal principles that apply in this proceeding.

26. I understand that a claim is anticipated, and therefore not novel, if every element of the claim is found in a single prior art reference. This means a prior art disclosure, such as a printed publication or another patent, must teach the invention with enough precision and detail that demonstrates that the subject matter existed in

the prior art. An invention is anticipated if a person of ordinary skill in the art would recognize that the prior art reference discloses all elements of the claim either explicitly or by necessary implication.

27. I understand that in an *inter partes* review, the petitioner shall have the burden of proving a proposition of unpatentability, including a proposition of anticipation, by a preponderance of the evidence.

V. OVERVIEW OF THE '080 PATENT

28. The '080 Patent deals with construction of software programs that distribute processing over multiple processors and multiple memories, as opposed to the use of a multiple processors and a single memory. The Patent refers to such programs as “parallel distributed programs.” EX1001, 1:25-30. Due to the inventors’ prior art publication, as shown below, all of this was well known long before the Patent was filed.

29. The Patent also uses the phrase “navigation programming,” which “includes the programming and use of self-migrating threads or tasks, which are threads or tasks that can migrate from one processor to another.” EX1001, 1:14-18. “Navigation” in this context refers to execution threads moving to the processor at which they are needed.

30. The Patent disclose that one feature of “navigational programming” is what it calls “distributed parallel computing,” or “DPC,” which is computing using

multiple concurrent distributed sequential computing “DSC” programs. EX1001, 5:57-59.

31. The Patent explains that a DSC program “injects” or “spawns” a separate DSC program that performs “fetching” of data. *Id.*, 6:43-47. “Fetch” was long a common term in the art for retrieving data from a memory.

32. A DPC program is a combination of a DSC program and a fetching DSC program:

The computing DSC program and the fetching DSC program together make a DPC program. The two DSC threads of the DPC program run concurrently to perform parallel or pipelined computing. In this particular example, the computations and the fetching are pipelined.

Id., 6:61-65

33. The Patent defines “strong mobility” as follows:

Navigational programming may be facilitated by mobile agent systems. Self-migrating threads may be in the form of mobile agents. Such a mobile agent has the ability to halt its execution, encapsulate the values of its variables, move to another node, restore the state, and continue executing. This ability is often referred to as strong mobility.

Id., 8:20-26

34. The Patent then points to the prior art MESSENGERS system as exhibiting strong mobility, and describe “two types of variables in MESSENGERS: agent variables and node variables.” *Id.*, 8:20-25; 9:1-2. Messengers take data with them when they migrate, by means of a “hop”:

A Messenger’s programmer may tell it to migrate using the navigational statements, such as hop(). A destination node’s logical address or a logical link between the source and the destination nodes can be used as the argument for the statements. When a Messenger hops, it takes the data in its agent variables with it to wherever it migrates.

Id., 9:10-15

35. Spawning (generating a new messenger) occurs in the Patent by executing an “Inject()” statement. Further, “Messengers may be injected by a program, which is part of the MESSENGERS system, or by another Messenger into any of the daemon nodes, and they may start creating new logical nodes and links on the current or any other daemons.” *Id.*, 8:39-43.

Injecting new Messengers is performed by a daemon: “There are four tasks for a daemon. First, to accept signals, such as UNIX signals, to inject Messengers. These signals are sent by the program minject or another messenger. In addition, a daemon also provides a function, the calling of which would result in the autonomous caller Messenger being sent to a destination daemon.”

Id., 8:50-58.

VI. LEVEL OF ORDINARY SKILL IN THE ART

36. I understand that the Board will be asked to characterize a person having ordinary skill in the art (“POSITA”) at the time the purported invention was made. To make that determination, it is my understanding that the Board considers the following factors: (i) the educational level of active workers in the field, (ii) the type of problems encountered in the art, (iii) the prior art solutions to those problems, the rapidity with which innovations are made, and (v) the sophistication of the technology in the art. It is my understanding that the importance of each factor varies from case to case.

37. The ’080 Patent defines its field of invention as follows:

The present invention relates generally to computer programming, and more particularly to systems and methods for parallel distributed programming.

EX1001, 1:13-15.

38. The specification of the Patent (“Specification”) discloses methods of distributing parallel processing across multiple memories and multiple processors using self-migrating threads. EX1001, 2:8-18. The Specification states:

Generally, a parallel distributed program is configured to operate across multiple processors/nodes and multiple memories. In one aspect of the invention, a parallel distributed

program includes at least one distributed shared variable located across the multiple memories and one or more distributed programs configured to operate across multiple processors.

EX1001, 2:9-16

That is, the field of the invention is parallel computing.

39. The claims are drawn to parallelizing sequential computer programs by spawning child sequential computer programs and establishing a distributed shared variable and .

40. In my opinion, in order to understand the Specification and make and use the claimed invention without undue experimentation as of May 2003, one of ordinary skill in the art of the '080 Patent would have had a bachelor's degree in computer science, computer engineering, electrical engineering, or related field, and at least two years of experience in computer networking and parallel computing, or a person with a master's degree in one of the foregoing and at least one year of experience in the aforementioned fields. Additional education could substitute for professional experience, and vice-versa.

41. I had at least the background of such a POSITA at least as early as May 21, 2003.

VII. CLAIM CONSTRUCTION

42. I understand that Patent Owner does not believe any claim construction is necessary in this proceeding, and I concur.

43. I have accorded all of the claim terms their plain and ordinary meaning to a POSITA.

VIII. BACKGROUND OF THE PRIOR ART

44. In this section, I discuss the state of the art prior to the Patent as of May 24, 2003. As a POSITA is presumed to have been familiar with all relevant prior art, a POSITA would have been familiar with the material in this section.

45. I find it incredible that the Patent issued. It is anticipated by EX1008, a 1998 article in a prominent journal by Fukuda, Bic, Dillencourt, and Merchant (“Fukuda ’98”). I note that Bic and Dillencourt are named inventors on the ’080 Patent, yet their paper Fukuda ’98 was not disclosed to the Patent Office during prosecution, and the Examiner did not locate it. In fact, six prior art papers authored by at least one of the named inventors were found by the Examiner but not disclosed in any IDS. Applicants did disclose six publications on which one or more of the Applicants was an author.

46. The ’080 Patent cites the MESSENGERS system as an example implementation of the invention. EX1001, 2:6-61; 8:26-10:17. In MESSENGERS, “applications are developed as collections of mobile agents, which may be referred

to as Messengers.” EX1001, 8:26-10:17. The MESSENGERS system was the subject of at least 37 prior art publications, most of which were not disclosed to the Patent Office¹.

A. The MESSENGERS Project

47. By at least 1996, the MESSENGERS Project at the University of California at Irvine was performing research and publishing papers on, in its own words,

a new programming paradigm, called Navigational Programming, for distributed systems based on the principles of autonomously migrating processes, called Messengers. Each Messenger is able to migrate among nodes of a local area network using explicit hop statements, which support strong migration. Unlike mobile agents used for a variety of services on the Internet, the MESSENGERS system is intended for general-purpose scientific computing.

Id.

48. The staff of this project included all of the named inventors of the '080 Patent. *Id.* The project produced at least 37 papers through 2002 relating to the subject matter of the later-filed '080 Patent. *Id.* In 2003, it appears, a subset of the

¹ See, “Project: MESSENGERS AND NAVIGATIONAL PROGRAMMING,” available at <https://ics.uci.edu/~lbic/messengers/>.

MESSENGERS project staff decided to file a patent application claiming what they had already been disclosed to the public at least five years earlier.

49. The inventors of the '080 Patent and various authors of the prior art publications discussed below are academically related. Inventors Bic and Dillencourt were faculty members at the University of California at Irvine ("UCI"), and directed the MESSENGERS project. Inventor Pan received his Ph.D. at UCI for his work on MESSENGERS. Munehiro Fukuda and Fehmina Merchant also received their Ph.D.'s at UCI for work of MESSENGERS. Christian Wicke was a visiting student who worked on MESSENGERS.

B. Fukuda '96

50. In 1996, '080 inventors Bic and Dillencourt (along with Fukuda and Merchant), published "Intra- and Inter-Object Coordination with MESSENGERS," First Int'l Conf. on Coordination Models and Languages (COORDINATION '96), Cesena, Italy. It disclosed the architecture of MESSENGERS six years before the asserted priority date of the '080 Patent.

51. Fukuda '96 was not considered in the '080 prosecution

C. Fukuda '96b

52. In August 1996, '080 inventors Bic and Dillencourt (along with Fukuda) published a UCI Technical Report entitled, "MESSENGERS: a distributed computing environment for autonomous objects." It discloses:

In this paper we describe MESSENGERS, an autonomous-object-based system developed at the University of California, Irvine, which aims at distributed parallel computing. An autonomous object called a Messenger in our system, carries a complete program (referred to as its script), together with its current status information, including a program counter and local variables. This defines the Messenger's behavior.

Fukuda '96b, 3.

53. It discloses that Messengers are threads:

Hence, the amount of information necessary to manage a Messenger is comparable to a thread, rather than a Unix process.

54. Fukuda '96b was not considered in the '080 prosecution.

D. Fukuda '97a

55. In March 1997, '080 inventors Bic and Dillencourt (along with Fukuda) published "Performance of the MESSENGERS Autonomous-Objects-Based System," First Int'l Conf. on Worldwide Computing and Its Applications '97 (WWCA97), Tsukuba, Japan, March 10-11, 1997 (LNCS 1274, Springer-Verlag).

56. It discloses the architecture of MESSENGERS and the network environment in which it is used²:

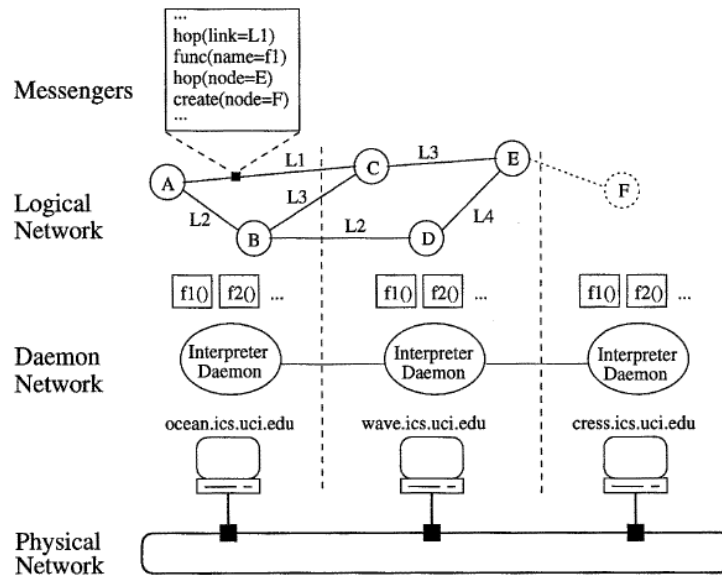


Fig. 1. Network Layers

57. Fukuda '97a was not considered in the '080 prosecution.

E. Fukuda '97b

58. In May 1997, '080 inventors Bic and Dillencourt (along with Fukuda and Merchant) published "Messages versus Messengers in Distributed Programming," Int'l. Conf. on Distributed Computing Systems (ICDCS-97), Baltimore, MD. It discloses navigational programming and illustrates how it differs

² By coincidence, Fukuda '97a references a book I co-authored concerning a problem in computational geometry that is used as a parallelization example in the paper.

from message-passing and shows its advantages. It also categorizes Messengers as mobile agents.

59. A version of Fukuda '97b was considered in the '080 prosecution.

F. Fukuda '98 (EX1008)

60. In July 1998, two of the three named inventors of the '080 Patent (Bic and Dillencourt) published a paper entitled, "Distributed Coordination with MESSENGERS" on pp. 291-311 of the journal *Science of Computer Programming* (1998). It describes the same MESSENGERS system that is referred to in the '080 Patent and discloses each limitation of each of the claims (1-18) of the '080 Patent, as shown in detail below. Fukuda '98 was not considered by the Patent Office in the '080 Prosecution.

61. Fukuda '98 describes a parallelization technique called MESSENGERS, which it elevates to the level of a programming "paradigm":

MESSENGERS is a paradigm for the programming of distributed systems. It is based on the principles of autonomous messages, called Messengers, which carry their own behavior in the form of a program. This enables them to navigate freely in the underlying computational network, communicate with one another, and invoke compiled node-resident C functions in the nodes they visit. Hence, a distributed application is viewed as a collection of C functions whose invocation and interoperation is orchestrated by Messengers.

EX1008, 291.

62. In Fukuda '98 (and also in other MESSENGERS publications), “MESSENGERS” in all uppercase refers to an entire system, while “Messengers” with initial caps only refers to individual processes.

63. MESSENGERS in Fukuda '98 is based on a mechanism it calls “medium-based coordination,” in which a common medium or state space, is shared by the processes. EX1008, 292. The shared “state space contains both functions and data.” *Id.* Messenger “processes may read or write data and dynamically incorporate functions into their behavior.” *Id.* “The shared space is structured by being mapped onto a logical network.” *Id.* “Every data item or function in the shared state space resides at some node in this network.” *Id.*

64. “Messengers have state information, consisting of local variables and control data and migrate to nodes containing data that they wish to read, write, or modify, and migrate to nodes containing functions that they wish to invoke.” EX1008, 292.

65. In particular, a Messenger is described as “a self-contained object, consisting of a program and current state, including a program counter and local variables.” EX1008, 293. “The program, which is a combination of interpreted MESSENGERS code and native C code, describes the Messenger’s behavior,

which allows it to navigate autonomously through the underlying communication network and carry out computation in the nodes it visits.” *Id.*

66. The system of Fukuda '98 is “implemented as a collection of daemons instantiated on all physical nodes participating in the distributed computation. The daemon’s task is to continuously receive Messengers injected from the user interface or arriving from other daemons, to interpret the Messengers’ behaviors described by their scripts, and to send them on to their next destinations as appropriate.” EX1008, 294.

67. The network in MESSENGERS is a logical network that includes both logical and physical nodes. EX1008 at 294. The logical network is “an application-specific computation network created on top of the daemon network. Multiple logical network nodes may be created on the same daemon network nodes, thus running on the same physical node, and they may be interconnected by logical links into an arbitrary topology.” *Id.*

68. “MESSENGERS distinguishes three types of variables: *messenger*, *node*, and *network* variables.” EX1008, 301. “Messenger variables are private to and carried by each Messenger as it propagates through the logical computational network. Node variables are resident in nodes and shared by all Messengers currently running on the same logical node. Network variables are predefined at

each logical node and give each Messenger access to the network information local to the current node.” *Id.*

69. “The general organization of the MESSENGERS interpreter is as follows. It alternates between two phases: interpretation of Messengers code and native-mode function execution. During the first phase, it makes a complete pass through the ready queue of Messengers and continues interpreting each until one of the following occurs: the Messenger terminates; the Messenger executes a navigational statement causing it to move to another node; or the Messenger invokes a compiled C function using the *func()* or *exec()* statements.” EX1008, 295.

70. Fukuda '98 explains *func()* as follows:

func(file =filename; name =func_name [; in = argument] [; out = argument]): Similar to exec, the parameter filename is a Unix path name specifying a file. This must contain one or more executable C functions (but not necessarily a complete program with a main() function). The function specified by the parameter func_name is invoked and the arguments specified by in are passed to it. Unlike with exec, the function executes as part of the current Messenger, that it, as part of the MESSENGERS interpreter process. The invoking Messenger is blocked until the function terminates and returns its results to the Messenger script via the out argument.

EX1008, 302

71. Further, “In MESSENGERS, [process creation] is done by injecting a new Messenger.” EX1008, 307-308.

72. Fukuda '98 give an example of a biological simulation using Messengers that “illustrates how a distributed simulation can be structured as a collection of functions, residing on different nodes of a network, where the invocation of these functions and the exchange of data among them is orchestrated by Messengers.” EX1008, 303. Fukuda '98 describes creating (“injecting”) a Messenger: “The corresponding Messenger is injected into the lung node, where it first invokes the *new_lung_cone* function (line 4) to compute the new styrene concentration using *inhaled_sty* and *c_veinal* (initially zero).” *Id.*, 304.

73. Fukuda '98 explains that a benefit of Messengers is that they are capable of “self-migration,” “i.e., autonomous navigation through both the logical and the physical networks.” EX1008, 306. Thus, “[a]n important implication of the MESSENGERS paradigm is that the individual entities move not only within the simulated space but actually migrate among processors in the network,” which occurs “when an entity crosses the boundary of a logical region” and “[t]he corresponding Messenger performs a *hop* statement, which causes it to automatically migrate into the processor responsible for the corresponding logical region.” *Id.*

G. Wicke '98

74. In May 1998, '080 inventors Bic and Dillencourt (along with Wicke and Fukuda) published "Automatic State Capture of Self-Migrating Computations in MESSENGERS," ICSE98 Int'l Workshop on Computing and Communication in the Presence of Mobility, (Part of Int'l Conference on Software Engineering), Kyoto, Japan. It discloses:

Self-migrating computations, commonly also referred to as mobile agents, are self-contained entities that can navigate autonomously through a network and perform various tasks in the nodes they visit. The underlying computational model is essentially a multi-threaded environment, augmented by navigational capabilities. Individual agents consist of a program and a state and are able to communicate with one another via shared or distributed memory mechanisms.

75. Wicke '98 was not considered in the '080 prosecution.

H. Fukuda '01 (EX1009)

76. Fukuda, Bic, Dillencourt, and Merchant, "MESSENGERS: Distributed Programming Using Mobile Agents," was published in *Transactions of the Society for Design and Process Science* (SDPS), Vol. 5, No. 4, Dec. 2001 ("Fukuda '01"). Fukuda '01 was not considered in the '080 prosecution.

77. It discloses distributed applications, defined as "collections of multiple concurrent activities executing on a loosely coupled network of computer nodes." EX1009, 95. It also explains the difference between explicit concurrency

and implicit concurrency in distributed applications: “The majority of distributed applications are developed using explicit concurrency, that is, using language where the programmer is required to divide the problem into separate activities and specify their dependencies. This is accomplished using various constructs, such as *fork*, *cobegin*, *parfor*, or concurrent process declarations. Implicit concurrency relies on the compiler to extract the potentially parallel activities and to provide both the mapping and the coordination primitives.” *Id.*

78. As in Fukuda '98, Fukuda '01 describes the MESSENGERS paradigm:

In this paper we present and explore a new programming paradigm for distributed systems, which differs from those mentioned above in its basic philosophy. The main concept is mobile agents, which are self-contained autonomous programs that have their own identity and behavior. Mobile agents can navigate through the underlying computational network, perform various tasks at the nodes they visit, and coordinate their activities with other agents participating in the distributed computation. We have developed a mobile agent system, called MESSENGERS, which we will use to illustrate concepts underlying general purpose distributed programming using mobile agents.

Ex.1009, 96

79. Fukuda '01 goes on to describe three levels of networks in MESSENGERS: the physical network, the daemon network, and the logical network (EX1009, 96-97), shown in its Figure 1:

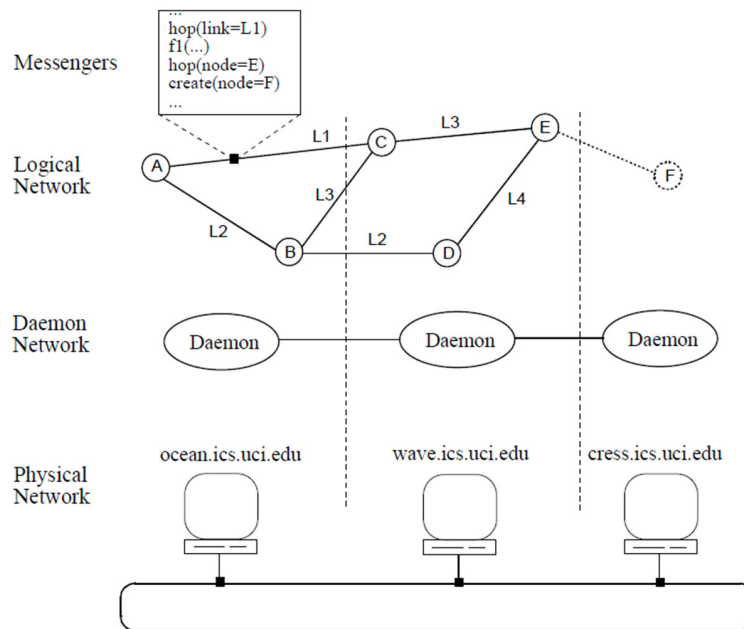


Fig. 1 Network Layers.

EX1009, 97.

80. Fukuda '01 also explains the principles of operation and the programming of the MESSENGER system. EX1009, 96-99. For example, the “[t]ask control/coordination statements,” including the “inject” statement, which “initiates (injects) the execution of one or more instances of another Messenger. The parameters specify the Messenger script to be injected and the initial parameters to be passed to the newly created Messenger. In addition the starting destination(s) may be specified.” *Id.*, 99.

81. Fukuda '01 was considered during the '080 prosecution.

I. Pan '01 (EX1005)

82. In September 2001, Pan and two of the three named inventors of the '080 Patent (Bic and Dillencourt) published "Distributed Sequential Computing Using Mobile Code: Moving Computation to Data," Int'l. Conf. on Parallel Processing (ICPP-01), Valencia, Spain.

83. It confirms that MESSENGERS is a mobile agent system: "In this section, we provide a very simple example, namely sequential matrix multiplication, to demonstrate why mobile agent systems such as MESSENGERS are among these useful tools." Pan '01, 79.

84. Pan '01 was considered during the '080 prosecution.

J. Pan '02

85. In October 2002, Pan and one of the three named inventors of the '080 Patent (Bic) published "Distributed Parallel Computing Using Navigational Programming: Orchestrating Computations Around Data." It discloses the "mobile agent system used in the MESSENGERS system developed at University of California Irvine" and how it is used for distributed parallel computing. It discloses distributed shared memory and "agent threads" that are "able to jump across machine boundaries.

86. It also explains the difference between explicit and implicit

navigation:

1. data distribution: The programmer must develop an application-dependent strategy for data distribution, and then construct, as by-products of the strategy, the data-processor map, and data-global-local map. These maps are then explicitly used in the code. In contrast, none of these maps shows up explicitly in DSM code, because the mapping is taken care of by the underlying DSM system. However, this by no means says that DSM programmers are home free. In fact, as shown in our Cholesky example, for the purposes of, say load balancing, a DSM programmer does need a data accessing strategy, from which constructing the maps is only a small step further.

2. mobile agent migration: With Navigational Programming, this is done explicitly using hop() statements, together with explicit data load/unload to/from agent variables. Variable sharing and communication is accomplished by execution of these inserted hop() and load statements. A DSM system provides all these implicitly for DSM programs. However, this seeming advantage can be the cause of poor performance and scalability of DSM programs because a programmer can easily make a wrong decision to move larger amount of data than needed. In this sense, Navigational Programming provides balanced convenience and flexibility: the code is easy to develop from the original algorithm, and the programmer has full control over exactly when and where data movement happens. Navigational

Programming allows the programmer to bridge distributed memory at the application level: by using the aforementioned two maps and the hop() statement, the programmer can view any shared variable allocated in distributed memory as if it were in contiguously shared memory with no boundaries.

Pan '02, 7.

87. Pan '02 was considered during the '080 prosecution

K. Art Raised During Prosecution

88. The Examiner used three of the above references as a basis for rejections: Pan (also “Pan” in the prosecution history, EX1005), Suzuki (also “Suzuki” in the prosecution history, EX1006) and Fukuda '97 (“Fukuda” in the prosecution history).

89. Applicants responded in an Amendment dated September 2, 2009 that “the combination of Pan, Suzuki and Fukuda fails to teach or suggest” limitations 1[c] and 1[d] of now-issued claim 1 and limitations 9[b], 9[c] and 9[d] of now-issued claim 9 (then-pending claim 15). EX1002, 59. Applicants wrote:

Pan, Suzuki and Fukuda simply do not teach or suggest transforming a distributed sequential computing (DSC) program into a distributed parallel computing program (DPC) program or a DSC program configured to transform into a DPC program wherein the DPC concurrently uses the DSC and child DSC to perform parallel and/or pipeline processing. Although Pan may describe the concept of DSC programs, which is computing using

a single locus of computation over distributed data (Application [0024]), Pan does not disclose or suggest transforming the DSC into a DPC by spawning a child DSC or a DSC configured to transform into a DPC by spawning a child DSC. Similarly, Suzuki describes the use of a collection of self-migrating threads. Although, as Applicants acknowledge at paragraph 0024 of the application, a self-migrating thread may be utilized to perform DSC, a description of the use of a collection of self-migrating threads does not disclose or suggest transforming a DSC into a DPC by spawning a child DSC or a DSC configured to transform into a DPC by spawning a child DSC.

Id.

90. Applicants continued:

As Pan notes at page 78, column 1, "Our implementations use MESSENGERS, a system for agent-based distributed computing developed in Information and Computer Science at the University of California, Irvine ([1, 5])." Reference 5 in Pan is Fukuda. Thus the combined teaching of Pan and Fukuda is effectively Pan, i.e., distributed sequential computing using MESSENGERS. Moreover, in light of Pan's discussion of implementation of DSCs using MESSENGERS, the fact that Fukuda describes replication does not teach or suggest transforming a DSC into a DPC by spawning a child DSC or a DSC configured to transform into a DPC by spawning a child DSC wherein the DPC concurrently uses the DSC and child DSC to perform parallel processing and/or operations, but rather only

suggests to one skilled in the art the replication of a MESSENGER within an individual DSC program. Thus, Fukuda does not cure the deficiency of Pan and Suzuki noted above, and, as a result, Pan, Suzuki and Fukuda can not establish a prima facie case of obviousness.

Id., 60

91. Applicants thus distinguished over the prior art on the sole basis that transforming a distributed sequential computing (DSC) program into a distributed parallel computing program (DPC) program or a DSC program configured to transform into a DPC program where the DPC concurrently uses the DSC and child DSC to perform parallel and/or pipeline processing was not disclosed.

92. Because Fukuda '98 remedies these supposed deficiencies, Fukuda '98 is not cumulative of the prior art considered during prosecution.

L. Conclusions Regarding the Prior Art

93. Every disclosure and claim limitation in the '080 Patent was described in, and/or at least obvious over, one of more of the prior art items described above. There is nothing novel about the '080 Patent, which ignores thousands of prior art publications on parallel processing.

94. Simply due to the prior publications of the inventors, all of the claimed matter in the '080 Patent would have already been known to a POSITA.

IX. THE CHALLENGED PATENT

95. I understand that Petitioner is challenging all claims (1-18) of the '080 Patent as anticipated by Fukuda '98. Of the 18 claims, claim 1 and 9 are independent. Claims 2-8 and 17 depend, directly or indirectly, from claim 1. Claim 9 is a system claim to a systems that essentially performs the steps of claim 1. Claims 10-16 and 18 depend, directly or indirectly, from claim 8. Claims 17 and 18 are identical, except that they depend from claims 1 and 9, respectively.

96. The Patent asserts that it discloses:

An approach to developing sequential or parallel applications in accordance with a preferred embodiment of the present invention, referred to as “navigational programming,” includes the programming and use of self-migrating threads or tasks, Which are threads or tasks that can migrate from one processor to another. These self-migrating threads or tasks may be in the form of mobile agents with strong mobility.

EX1001, 3:13-19

97. The Patent purports to claim this approach through the following steps in claim 1:

98. [1pre] “A method of developing a distributed parallel computing program, comprising steps of:”

99. The preamble was long known and conventional, as I showed above in the Background of the Prior Art section. It was conventional to develop a distributed parallel computing program.

100. [1a] “establishing at least one distributed shared variable, wherein the at least one distributed shared variable is a single variable that includes several variables that may be physically distributed across multiple memories.” A distributed shared variable that includes several variables that may be physically distributed across multiple memories was previously disclosed in numerous sources.

101. [1b] “developing at least one distributed sequential computing program to access the at least one distributed shared variable;” A goal of parallel computing is to divide a task into a set of individual sequential programs that can run simultaneously (“in parallel”). There is no point in establishing a distributed shared variable unless a sequential program is going to access it.

102. [1c] “transforming the at least one distributed sequential computing program into at least one distributed parallel computing program by spawning at least one child distributed sequential computing program from the at least one distributed sequential computing program when at least one intermediate condition occurs within the at least one distributed sequential computing program.” The spawning of child sequential programs is a fundamental operation in the process of parallelization. In the claim, the spawning occurs when a distributed program needs

to pass off a computing chore to a child process. Recognition of this need is the claimed “intermediate condition.”

103. [1d] “wherein the at least one distributed parallel computing program concurrently uses the at least one distributed sequential computing program and the at least one spawned child distributed sequential computing program to perform parallel processing and/or operations.” This simply expresses that computation is to take place is parallel, which was known since the first child distributed sequential computing programs were spawned. The entire purpose of spawning a new sequential program is to permit parallel processing.

104. [1e] “wherein the at least one intermediate condition comprising one intermediate result that will be required by the at least one spawned child distributed sequential computing program to continue computation.” This limitation is ungrammatical as written, and I believe a POSITA would understand that “comprising” should be understood as “comprises.”

105. Claim 2 depends from claim 1 and adds “wherein each of the at least one distributed sequential computing program includes at least one self-migrating thread, capable of migrating among at least one processor and capable of accessing the at least one distributed shared variable.” Using self-migrating threads was known at least as early as 1999, and is fully disclosed in Suzuki (EX1006).

106. Claim 3 depends from claim 1, but adds “wherein the at least one distributed shared variable is configured to be located among a plurality of memories.” That is the very definition of a “distributed shared variable.” If the variable is located in only one memory, it may be a “shared variable,” but is not a “distributed shared variable.”

107. Claim 4 depends from claim 1 and adds “wherein the distributed parallel computing program is configured to operate across multiple processors.” This is simply part of the definition of “distributed parallel computing program.” A computing program that runs on only one processor is not “distributed.”

108. Claim 5 depends from claim 4, and adds “wherein the distributed parallel computing program is configured to operate using multiple threads.” Multiple threading was a conventional programming technique. In the context of distributed parallel computing it was amply disclosed, e.g., in Suzuki (EX1006).

109. Claim 6 depends from claim 1, and adds “wherein the distributed parallel computing program is configured to operate across multiple nodes.” This is true of all network-distributed parallel computing programs.

110. Claim 7 depends from claim 1, and adds “further comprising the step of configuring the at least one distributed sequential computing program to include at least one mobile agent.” It was known, e.g., in Fukuda '98, that a sequential computing program could include a mobile agent.

111. Claim 8 depends from claim 1 and adds “wherein the at least one mobile agent is implemented using self-migrating threads.”

112. Claim 9 is an independent claim to a system for performing the method of claim 1.

113. Claim 10 depends from claim 9 and adds “wherein the at least one distributed sequential computing program comprises at least one mobile agent,” making it effectively the same as claim 7.

114. Claim 11 depends from claim 10 and adds “wherein the at least one mobile agent is implemented using self-migrating threads.” Implementing mobile agents via self-migrating threads was well-known. An entire prior art paper, Suzuki (EX1006) was devoted to this topic. ’080 inventors Bic and Dillencourt were co-authors on Suzuki.

115. Claim 12 depends from claim 10 and adds “wherein the at least one mobile agent is an explicit-navigation mobile agent.” An “explicit-navigation mobile agent is one that is pre-programmed to migrate, e.g., with “migration statements.” See EX1001, 3:13-22. The difference between explicit and implicit navigation was disclosed, e.g., in Fukuda ’96:

The function duplicate() allows a new Messenger to be created by cloning. It is provided mainly for performance reasons, since it replicates an existing behavior and hence does not involve the file system. The creation of Messengers using this function

should not be confused with the replication of Messengers during navigation. The latter is implicit and is the result of the Messenger's following multiple logical links during a hop() statement. The function duplicate(), on the other hand, causes an explicit creation of a new Messengers inside a node.

Fukuda '96, 185

116. Similarly in Fukuda '96b:

The injection of Messengers using these library functions should not be confused with the automatic replication of Messengers during navigation. The latter is implicit and is the result of the Messenger's following multiple logical links during a hop statement. The functions m_inject, m_duplicate, and m_fork, on the other hand, cause an explicit creation of a new Messenger inside a node.

Fukuda '96b, 11

117. Claim 13 depends from claim 10 and adds “wherein the at least one mobile agent is an implicit-navigation mobile agent.” An “implicit-navigation mobile agent is one that is not explicitly pre-programmed to migrate, but decides itself when and where to migrate, possibly based on the data to be processed. See EX1001, 3:13-22 and the discussion of claim 12 in the paragraphs immediately above.

118. Claim 14 depends from claim 10 and adds “wherein the at least one mobile agent is configured to move from one processor to another processor.” That is the very meaning of the word “mobile” in the context of the ’080 Patent.

119. Claim 15 depends from claim 10 and adds “wherein the at least one mobile agent operates within a distributed shared memory system.” The prior art MESSENGERS system was well known to be a “distributed shared memory system” utilizing mobile agents.

120. Claim 16 depends from claim 9 and adds “wherein the at least one processor is located across a physical network.” The prior art MESSENGERS system was well known to operate across a physical network.

121. Claim 17 depends from claim 1 and adds “wherein the distributed parallel computing program is configured to operate across a distributed shared memory system.” The prior art MESSENGERS system was well known to be a distributed parallel computing program is configured to operate across a distributed shared memory system.

122. Claim 18 depends from claim 9 and adds “wherein the distributed parallel computing program is configured to operate across a distributed shared memory system,” which is the same limitation as in claim 17.

X. GROUND 1: CLAIMS 1-18 ARE ANTICIPATED BY FUKUDA ’98 (EX1008)

A. Claim 1

123. Claim 1 as a whole and each limitation of claim 1 is disclosed in Fukuda '98, as shown below.

1. **[1pre] A method of developing a distributed parallel computing program, comprising steps of:**

124. Whether or not the preamble is limiting, it is disclosed by Fukuda '98, which discloses a method of developing a distributed parallel computing program.

2. **[1a] establishing at least one distributed shared variable, wherein the at least one distributed shared variable is a single variable that includes several variables that may be physically distributed across multiple memories;**

125. The term “distributed shared variable” is explained in the '080 Patent:

In one aspect of navigational programming, a distributed shared variable (“DSV”) may be used. DSVs and self-migration make shared variable programming possible beyond shared memory that provides single-address memory space. A DSV is logically a single variable that includes several variables that may be physically distributed across multiple memories. An example of a DSV is a global array.

EX1001, 3:33-39

126. Specifically regarding MESSENGERS, the '080 Patent describes “node variables”:

There are two types of variables in MESSENGERS: agent variables and node variables. An agent variable is private to a particular Messenger and travels with that Messenger as it migrates through the logical network. A node variable is

stationary and is accessible by all Messengers currently at the logical node to which the variable belongs. Hence agent variables can be used to carry data between nodes, while node variables can be used for inter-agent communication

EX1001, 9:1-9.

127. Fukuda '98 discloses the claimed “distributed shared variables”:

“Node variables are resident in nodes and shared by all.” EX1008, 301.

“Messengers must generally interact with one another in both time and space.

Spatial interaction is accomplished via shared node variables defined in every logical node. Every Messenger script contains a node variables declaration that uses the same style and conventions as C.” *Id.*, 302. The C language permits a variable (e.g., a structure) to contain other variables.

128. Further, “[t]o support temporal interaction, MESSENGERS provide support for *virtual time*, i.e., a global time line consistent over all physical nodes [16]. Conceptually, this is a globally shared variable that simulates the passage of time. It allows any Messenger to suspend itself until a certain point in the virtual time has been reached.” *Id.* “To ensure that all Messengers have arrived at the lung node before the next cycle begins, the shared node variable *n_msgtrs* in the lung node is used as a synchronization counter.” *Id.*, 304.

129. Fukuda '98 explains that nodes (and therefore node variables, which reside at nodes) in MESSENGERS are physically distributed across multiple

memories. MESSENGERS includes a physical network (a LAN or WAN).

EX1008, 294. Messengers (which interact with node variables at nodes) “move not only within the simulated space but actually migrate among processors in the network.” EX1008, 306. A POSITA would have understood that a LAN (local area network) and a WAN (wide area network) each require physically distinct memories and processors at each node and therefore Fukuda '98 necessarily discloses that the node variables are physically distributed across multiple memories.

130. Fukuda '98 describes the architecture of MESSENGERS as comprising “three levels of networks”:

1. Physical network: This is the lowest level network (a LAN or WAN), which constitutes the underlying computational resource.

2. Daemon network: This network is superimposed over the physical layer. Each daemon node is a UNIX process running a MESSENGERS language interpreter. The daemon network topology is described by the user at system's initialization. This involves selecting an arbitrary subset of the physical nodes to run the daemon process on (which may be done automatically by the system based on current loads) and specifying arbitrary inter-daemon links. Each such link is mapped onto a path of one or more links in the physical network and is used to guide the mapping of logical links and nodes at run time. The main purpose of the daemon network is to provide a virtual computational

resource, thus isolating the applications from the details of the physical network topology.

3. Logical network: This is an application-specific computation network created on top of the daemon network. Multiple logical network nodes may be created on the same daemon network nodes, thus running on the same physical node, and they may be interconnected by logical links into an arbitrary topology. It is this network that is used by Messengers when navigating through the system.

131. Therefore, Fukuda '98 discloses limitation 1[a].

3. [1b] developing at least one distributed sequential computing program to access the at least one distributed shared variable;

132. I believe a POSITA would understand “developing” to mean “creating.” The '080 Patent describes “distributed sequential programming” as “computing using a single locus of computation over distributed data, possibly stored in a DSV. A self-migrating thread may be utilized to perform DSC.”

EX1001, 3:50-54. The '080 Patent gives an example of a sequential program that performs matrix multiplication. EX1001, 6:16-40. Matrix multiplication is an example of a computational problem that can benefit from parallelism because the dot products of rows and columns can be computed independently in parallel, yet matrix elements must be shared across processes.

133. Fukuda '98 discloses “developing at least one distributed sequential computing program to access the at least one distributed shared variable” of limitation [1b]. Fukuda '98 gives a “coordination example” of a biological simulation of the circulatory system: “We illustrate how a distributed simulation can be structured as a collection of functions, residing on different nodes of a network, where the invocation of these functions and the exchange of data among them is orchestrated by Messengers.” EX1008, 303.

134. Fukuda '98 describes this distributed sequential computing program:

Our implementation uses MESSENGERS as a control language to coordinate the operation and interaction of compiled node-resident C functions, which carry out the actual computations of the model. The basic approach is to map each organ onto a separate node, which contains the necessary sets of constants and differential equations as C functions. The toxin-carrying fluids, such as blood, are implemented as waves of consecutive Messengers, which cycle through the organism along the predefined paths, thus mimicking the actual flow through the body over time. As they pass through the organs, they trigger the execution of the appropriate functions to compute the new concentrations and other values for the current simulated time increment.

EX1008, 304.

135. Fukuda '98 describes a Messenger script for styrene concentration at 304-305 and reproduces the script in Figure 4:

```
(1) styrene_sim(n_iter, inhaled_sty) {  
(2)   c_veinal == 0;  
(3)   for (i = 0; i < n_iter; i++) {  
(4)     func(name="new_lung_conc"; in=c_veinal, inhaled_sty; out = c_art);  
(5)     n_msgrs = number_of_organs;  
(6)     hop(link="arterial");  
(7)     func(name="new_organ_conc"; in=c_art; out=c_ven_o);  
(8)     hop(link="veinal");  
(9)     c_veinal += c_ven_o;  
(10)    if (--n_msgrs > 0)  
(11)      exit;  
(12)  }  
(13) }
```

Fig. 4. Messenger script for styrene inhalation.

136. In the above code, *inhaled_sty* and *c_veinal* are distributed shared variables.

137. Thus, Fukuda '98 (EX1008) discloses limitation [1b].

4. **[1c] transforming the at least one distributed sequential computing program into at least one distributed parallel computing program by spawning at least one child distributed sequential computing program from the at least one distributed sequential computing program when at least one intermediate condition occurs within the at least one distributed sequential computing program**

138. To clarify the meanings of the terms in this limitation, I refer to the '080 Specification.

139. The '080 Patent describes “transforming the at least one distributed sequential computing program into at least one distributed parallel computing program by spawning at least one child distributed sequential computing program from the at least one distributed sequential computing program when at least one

intermediate condition occurs within the at least one distributed sequential computing program.” It gives an example of a sequential program that performs matrix multiplication, and explains how, in a distributed environment, “we may use a processor to post-fetch the computed C sub-matrix and pre-fetch the next pair of A and B sub-matrices,” as shown in the DSC program below:

```
(1) for i = 0 to p - 1
(2)   for j = 0 to p - 1
(2.1)     inject (WR (i, j))
(2.2)     waitEvent (IOb(i,j))
(3)       Cij = Ai Bj
(3.1)     hop (!μ)
(4)   end for
(5) end for
```

EX1001, 6:33-40.

140. The '080 Patent explains the operation of this code:

WR() is a separate DSC program that performs fetching. It is “injected,” or spawned, by the computing thread, and it hops to the other node to perform fetching, after which it “signals” an event to indicate that the fetching is done. The computing thread will (2.2) wait for the event to make sure that the data is correctly post- and pre-fetched, (3) compute the sub-matrix multiplication, and (3.1) hop to the other node to perform the computation for the next loop.

141. The DSC program “injects” “WR()” using an “inject()” statement at line (2.1). EX1001, 6:36. The '080 Patent explains how “[t]he computing DSC

program and the fetching DSC program together make a DPC program. The two DSC threads of the DPC program run concurrently to perform parallel or pipelined computing. In this particular example, the computations and the fetching are pipelined.” EX1001, 6:61-65.

142. The '080 Patent later explains that, in MESSENGERS, a “Messenger can spawn another Messenger using a statement, such as inject(). Synchronization among Messengers uses ‘events,’ and statements, such as signalEvent() and waitEvent().” EX1001, 9:16-19; 8:38-43; 8:50-52.

143. The '080 Patent further explains what is meant in limitation 1[c] by “at least one intermediate condition occurs within the at least one distributed sequential computing program,” the '080 Patent describes each box of Figures 3(a), 4(a), and 5(a) as representing a computation, wherein “[a] box marked with R means the computation represented by the box produces intermediate result that will be required by the following computation on the next node, whereas a box marked with NR means the computation on the next node is independent of the computation represented by the box.” EX1001, 7:47-52. That is, the “intermediate condition” is the receipt of the intermediate result from the computation performed on the current node:

[T]he computation on the next node is scheduled as soon as the dependency condition allows it. That is, since in the DSC the computation on the next node only depends on the intermediate

result from part of the computation on the current node, as shown in FIG. 3(a), the thread can clone itself as soon as the computation of R is done, and have the clone hop to the next node carrying the intermediate result and continue the computation at the same time when it continues the computation of NR on the current node, as shown in FIG. 3(b)

Id.

144. With this understanding, Fukuda '98 discloses limitation 1[c]. It explains that “a Messenger script may include statements to dynamically link and invoke C functions, precompiled and executed in native-mode on the current node. This permits the behavior of each Messenger, and hence the functionality of the entire application, to be composed dynamically.” EX1008, 301.

145. As taught by Fukuda '98, two statements provide an interface to executable C functions via the Unix file system. EX1008, 301. The *exec()* statement contains a *main()* function “which is spawned as a separate Unix process, i.e., operating concurrently with the invoking Messenger.” *Id.* In the *exec()* function, shown below, “[t]he arguments specified by *in* are passed to the new process at the time of creation and results are returned to the invoking Messenger via the *out* argument”:

*exec(file = filename [; in = argument] [; out = argument;]
[nowait]).*

Ex.1008, 301.

146. The styrene distributed simulation example in Fukuda '98 discloses limitation [1c]. As Fukuda '98 explains, the “[t]he corresponding Messenger is injected into the lung node.” EX1008, 304 (emphasis added). As understood from the '080 Specification, “injecting” is the same as “spawning.” EX1001, 6:44-45. The “injecting” creates a process that did not previously exist, hence it is “spawned,” Therefore, “injecting” the Messenger into the lung node is “spawning at least one child distributed sequential computing program from the at least one distributed sequential computing program,” i.e., the distributed simulation is a distributed sequential computing program and the Messenger that is “injected” and runs the script shown in Figure 4 for the styrene inhalation is also a distributed sequential computing program , which, together, transforms the distributed simulation to a distributed parallel computing program. The intermediate condition occurs when, within the distributed simulation, each cycle begins, as described by Fukuda '98. EX1008 at 304-305.

147. Fukuda '98 explains that Messengers are programs: “MESSENGERS is a paradigm for the programming of distributed systems. It is based on the principles of autonomous messages, called Messengers, which carry their own behavior in the form of a program.” EX1008, 291.

148. Thus, consistent with the '080 Specification, Fukuda '98 discloses the transformation of at least one distributed sequential computing program into at least one distributed parallel computing program by spawning at least one child distributed sequential computing program from the at least one distributed sequential computing program (i.e., using, for example, the *exec()* statement which contains a *main()* function (EX1008, 301) or using the “inject” statement (Ex.1008 at 304)) when at least one intermediate condition occurs within the at least one distributed sequential computing program (e.g., the arguments specified by in, as disclosed in Fukuda '98 (EX1008, 301) or the parameters to be passed to a newly created Messenger when invoking the “inject” statement (EX1008, 304).

149. Therefore, Fukuda '98 discloses limitation [1c].

5. **[1d] wherein the at least one distributed parallel computing program concurrently uses the at least one distributed sequential computing program and the at least one spawned child distributed sequential computing program to perform parallel processing and/or operations,**

150. This was shown above in the discussion of [1d]. The spawning of a and running of a child process causes parallel processing to be performed.

151. As discussed, the '080 Patent explains that, when the separate DSC program, WR(), that performs fetching is “injected” or “spawned,” “[t]he computing DSC program and the fetching DSC program together make a DPC program. The two DSC threads of the DPC program run concurrently to perform

parallel or pipelined computing. In this particular example, the computations and the fetching are pipelined.” EX1001, 6:61-65.

152. Likewise, Fukuda '98 discloses that *exec()*, contains a *main()* function, “which is spawned as a separate Unix process, i.e., operating concurrently with the invoking Messenger.” EX1008, 301.

153. Fukuda '98 further discloses the distributed simulation of where Messengers are running concurrently, as shown in Figure 3 (e.g., the messengers for C_ven_f, C_ven_m, C_ven_r, and C_ven_l are shown to be running concurrently):

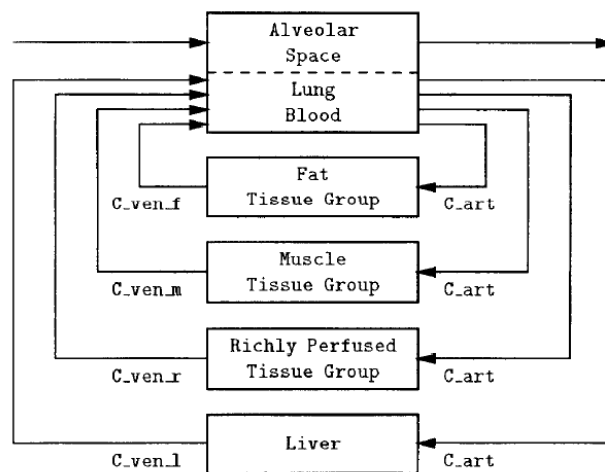


Fig. 3. A toxicology model.

Ex.1008 at 304.

154. Thus, Fukuda '98 discloses limitation [1d].

6. **[1e] wherein the at least one intermediate condition comprising one intermediate result that will be required by the at least one spawned child distributed sequential computing program to continue computation.**

155. As discussed above, the '080 Patent describes each box of Figures 3(a), 4(a), and 5(a) as representing a computation, wherein “[a] box marked with R means the computation represented by the box produces intermediate result that will be required by the following computation on the next node, whereas a box marked with NR means the computation on the next node is independent of the computation represented by the box.” EX1001, 7:47-52. In other words, the intermediate condition is the receipt of the intermediate result from part of the computation on the current node:

[T]he computation on the next node is scheduled as soon as the dependency condition allows it. That is, since in the DSC the computation on the next node only depends on the intermediate result from part of the computation on the current node, as shown in FIG. 3(a), the thread can clone itself as soon as the computation of R is done, and have the clone hop to the next node carrying the intermediate result and continue the computation at the same time when it continues the computation of NR on the current node, as shown in FIG. 3(b).

Ex.1001, 7:54-63.

156. As taught by Fukuda '98, two statements provide an interface to executable C functions via the Unix file system. EX1008, 301. The function *exec()* contains a *main()* function, “which is spawned as a separate Unix process, i.e., operating concurrently with the invoking Messenger.” *Id.*

157. In the *exec()* function, shown below, “[t]he arguments specified by *in* are passed to the new process at the time of creation and results are returned to the invoking Messenger via the *out* argument.”

```
exec(file = filename [; in = argument] [; out = argument;]  
[nowait])
```

EX1008, 301

158. The distributed simulation example in Fukuda '98 discloses limitation [1e]. As Fukuda '98 describes, the “[t]he corresponding Messenger is injected into the lung node.” EX1008, 304. The intermediate condition occurs when, within the distributed simulation, each cycle begins, as described by Fukuda '98, and ends when the *n_msgs* variable decrements. EX1008, 304-305.

159. Thus, consistent with the '080 Specification, Fukuda '98 discloses the at least one intermediate condition comprising one intermediate result that will be required by the at least one spawned child distributed sequential computing program to continue computation (e.g., using, for example, the *exec()* statement which contains a *main()* function) when at least one intermediate condition also comprises an intermediate result (e.g., the arguments specified by *out*). This also occurs when invoking the “inject” statement, as shown in the example in Fukuda '98, where the injected Messenger returns to the lung node after adding the

c_ven_o value to the cumulative node variable *c_veinal* and the shared variable *n_msgrs* in the lung node decrements. Ex.1008 at 304-305

160. Thus, Fukuda '98 discloses limitation [1e].

161. Therefore, claim 1 is anticipated by Fukuda '98.

B. Claim 2: The method of claim 1, wherein each of the at least one distributed sequential computing program includes at least one self-migrating thread, capable of migrating among at least one processor and capable of accessing the at least one distributed shared variable.

162. As shown in connection with claim 1, Fukuda '98 discloses claim 1.

163. The '080 Patent defines “self-migrating threads or tasks” as “threads or tasks that can migrate from one processor to another.” The limitation of claim 2, by using the phrase “capable of migrating among at least one processor,” appears to include the possibility of a self-migrating thread migrating within a single processor, which is not described in the specification and makes no sense in the context of “migration.” I believe a POSITA would understand “capable of migrating among at least one processor” to mean “capable of migrating among a plurality of processors,” and I have interpreted it that way.

164. Fukuda '98 discloses self-migrating threads as “Messengers,” which are “a self-contained object, consisting of a program and current state, including a program counter and local variables. The program, which is a combination of interpreted MESSENGERS code and native C code, describes the Messenger’s

behavior, which allows it to navigate autonomously through the underlying communication network and carry out computation in the nodes it visits.” EX1008, 293. Messengers are “autonomous messages,” “which carry their own behavior in the form of a program” (*Id.*, 291) and are “capable of self-migration, i.e., autonomous navigation through both the logical and the physical networks” (*Id.*, 306).

165. Fukuda '98 further discloses that the self-migration capability of Messengers means that “the individual entities move not only within the simulated space but actually migrate *among processors in the network*.” EX1008, 306 (emphasis added). “This occurs when an entity crosses the boundary of a logical region. The corresponding Messenger performs a *hop* statement, which causes it to automatically migrate into the processor responsible for the corresponding logical region.” *Id.*

166. Fukuda '98 further discloses that the Messengers are capable of accessing a distributed shared variable: “[n]ode variables are resident in nodes and shared by all Messengers currently running on the same logical node.” EX1008, 301. “[A] Messenger can read and write any of the node variables of the node in which it currently resides. Consequently, any collection of Messengers can interact with each other by hopping to a common node and reading/writing agreed-upon node variables.” EX1008, 302.

167. Fukuda '98 shows a Messenger accessing distributed variables *c_veinal* and *n_msgrs*: “The circulation is accomplished using the two *hop* statements; the first (line 6) replicates the Messenger to all organ nodes, where the corresponding *new-erg-cone* computes the new styrene concentration; the second *hop* statement (line 8) then takes the Messenger back to the lung node along the veinal links, where it adds its *c_ven_o* value (which represents the organ’s contribution to the incremental change in the veinal styrene concentration) to the cumulative node variable *c_veinal* (line 9). To ensure that all Messengers have arrived at the lung node before the next cycle begins, the shared node variable *n_msgrs* in the lung node is used as a synchronization counter.” EX1008, 304-305; see also 302, describing “virtual time” (i.e., “a globally shared variable that simulates the passage of time”).

168. Therefore, Fukuda '98 discloses claim 2.

C. **Claim 3: The method of claim 1, wherein the at least one distributed shared variable is configured to be located among a plurality of memories**

169. Fukuda '98 discloses claim 1, as discussed above.

170. The limitation of claim 3 follows directly from the phrase “distributed shared variable.” If the shared variable were not located among a plurality of memories, it might be a “shared variable,” but not a “distributed shared variable.” Therefore, claim 3 does not further limit claim 1.

171. Nevertheless, as discussed above in connection with limitation [1b], Fukuda '98 gives an example of a distributed shared variable, i.e., virtual time, which is “a global time line consistent over all physical nodes” which is “a globally shared variable that simulates the passage of time.” EX1008, 302. Because it is on “all physical nodes,” it is a “distributed shared variable.”

172. Further, because Messengers can migrate to nodes on different processors (as described at EX1008, 306), the global variable “*virtual time*” must be located among a plurality of memories.

173. Therefore, Fukuda '98 discloses claim 3.

D. Claim 4: The method of claim 1, wherein the distributed parallel computing program is configured to operate across multiple processors.

174. Fukuda '98 discloses claim 1, as discussed above.

175. As discussed above in connection with claim 2, the Messenger self-migrating threads migrate among different processors. Fukuda '98 discloses “the individual entities move not only within the simulated space but actually migrate among processors in the network.” EX1008, 306. “This occurs when an entity crosses the boundary of a logical region. The corresponding Messenger performs a *hop* statement, which causes it to automatically migrate into the processor responsible for the corresponding logical region.” *Id.*

176. Therefore, Fukuda '98 discloses claim 4.

E. Claim 5: The method of claim 1, wherein the distributed parallel computing program is configured to operate using multiple threads.

177. Fukuda '98 discloses claim 1, as discussed above.

178. Fukuda '98 discloses that the distributed parallel computing program is configured to operate using multiple Messengers (i.e., multiple threads). If the distributed shared variable *n_msgrs* is ever greater than one, then multiple threads are operating.

179. Messengers are “autonomous messages,” “which carry their own behavior in the form of a program” (EX1008, 291) and are “capable of self-migration, i.e., autonomous navigation through both the logical and the physical networks” (*Id.*, 306).

180. “To accomplish their mission, Messengers must generally interact with one another in both time and space. Spatial interaction is accomplished via shared node variables defined in every logical node. Every Messenger script contains a node variables declaration that uses the same style and conventions as C. This provides the necessary mapping between the Messenger code and the actual node-resident variables. Using the declared names, a Messenger can read and write any of the node variables of the node in which it currently resides. Consequently, any collection of Messengers can interact with each other by

hopping to a common node and reading/writing agreed-upon node variables.”

EX1008,.

181. Fukuda '98 gives an example that uses multiple Messengers, as shown in Figure 3 and described with respect to Figure 4: “To ensure that all Messengers have arrived at the lung node before the next cycle begins, the shared node variable *n_msgrs* in the lung node is used as a synchronization counter. At the beginning of each cycle it is set to the number of organs (line 5) thus counting the number of Messenger replicas.” Ex.1008, 304-305.

182. Therefore, Fukuda '98 discloses claim 5.

F. Claim 6: The method of claim 1, wherein the distributed parallel computing program is configured to operate across multiple nodes.

183. Fukuda '98 discloses claim 1, as discussed above.

184. Fukuda '98 discloses that operation across multiple nodes is part and parcel of the MESSENGERS architecture:

The MESSENGERS coordination model, the subject of this paper, is also based on an underlying shared state space through which processes communicate. The underlying state space contains both functions and data. Processes may read or write data and dynamically incorporate functions into their behavior. The shared space is structured by being mapped onto a logical network. Every data item or function in the shared state space resides at some node in this network. Individual processes, called

Messengers have state information, consisting of local variables and control data. Messengers are able to navigate freely and autonomously through the logical network. They migrate to nodes containing data that they wish to read, write, or modify, and to nodes containing functions that they wish to invoke.

EX1008, 292.

Messenger is a self-contained object, consisting of a program and current state, including a program counter and local variables. The program, which is a combination of interpreted MESSENGERS code and native C code, describes the Messenger's behavior, which allows it to navigate autonomously through the underlying communication network and carry out computation in the nodes it visits.

Id., 293.

185. Therefore, Fukuda '98 discloses claim 6.

G. **Claim 7: The method of claim 1, further comprising the step of configuring the at least one distributed sequential computing program to include at least one mobile agent.**

186. Fukuda '98 discloses claim 1, as discussed above.

187. Fukuda discloses the limitation of claim for the simple reason that the disclosed “self-migrating threads” are mobile agents. The '080 Patent itself admits that “These self-migrating threads or tasks may be in the form of mobile agents with strong mobility.” EX1001, 1:18-19. Further, “An example of such a system is

MESSENGERS, in which applications are developed as collections of mobile agents, which may be referred to as Messengers.” *Id.*, 8:29-31.

188. The same “self-migrating threads” are disclosed in Fukuda ’98. I also note that “mobile agents” is listed as a keyword on the first page of Fukuda ’98.

EX1008, 291. Further:

Notably, autonomous navigation is used by a number of systems to create “mobile agents” (see Section 4). While their primary purpose is to roam the Internet, MESSENGERS uses navigation as a primary structuring tool, which, in conjunction with dynamic functional composition, yields a powerful coordination paradigm for general-purpose distributed computing.

Id., 294.

189. Therefore, Fukuda ’98 discloses claim 7.

H. Claim 8: The method of claim 7, wherein the at least one mobile agent is implemented using self-migrating threads

190. Fukuda ’98 discloses claim 1, as discussed above.

191. As discussed in connection with claim 2, the Messengers (i.e., mobile agents) are “autonomous messages,” “which carry their own behavior in the form of a program” (EX1008, 291) and are “capable of self-migration, i.e., autonomous navigation through both the logical and the physical networks” (*Id.*, 306).

192. As discussed in connection with claim 7, mobile agents are implemented using self-migrating threads.

193. Therefore, Fukuda '98 discloses claim 8.

I. Claim 9

194. Claim 9 is effectively a system form of claim 1.

1. [1pre] A distributed parallel computing system, having at least one memory area and at least one processor, comprising:

195. Whether or not the preamble is limiting, it is disclosed by Fukuda '98. MESSENGERS is a distributed parallel computing system. "MESSENGERS is a paradigm for the programming of distributed systems." EX1008. Abstract. Because the Messengers operate on different processors simultaneously, MESSENGERS is a distributed parallel computing system.

196. Fukuda '98 discloses at least one memory area: "In MESSENGERS. the programmer's view is navigational, concerned with how to move through the logical network, and when to interact with the data or functions stored at a node." *Id.*, 293. Data or functions cannot be stored at a node unless the node has a memory area. A function cannot be executed at a node unless the node has a processor.

197. Further, "An important implication of the MESSENGERS paradigm is that the individual entities move not only within the simulated space but actually migrate among processors in the network." *Id.*, 306.

198. Therefore, Fukuda '98 discloses [9pre].

2. **[9a] at least one distributed shared variable capable of loading into the at least one memory area, wherein the at least one distributed shared variable is a single variable that includes several variables that may be physically loaded into the at least one memory area;**

199. Limitation [9a] is essentially the same as limitation [1a], but with the addition of references to “loading.” Hence see [1a]. A variable cannot be “physically distributed across multiple memories, as in [1a] unless it is “loaded” into those memories.

200. Therefore, Fukuda '98 discloses [9a].

3. **[9b] at least one distributed sequential computing program, configured to operate in the at least one processor, configured to access the at least one distributed shared variable,**

201. Limitation [9b] is essentially the same as limitation [1b], but with the addition of reference to a “processor.” Hence see [1b]. A distributed shared variable (as in [1b]) cannot be accessed unless the distributed sequential computing program (as in [1b]) is configured to operate in a processor.

202. Therefore, Fukuda '98 discloses [9b].

4. **[9c] configured to transform into at least one distributed parallel computing program by spawning at least one child distributed sequential computing system program when at least one intermediate condition occurs within the at least one distributed sequential program,**

203. Limitation [9c] is identical to limitation [1c], except that the phrase “transforming the at least one distributed sequential computing program” has been

replaced by “configured to transform.” This is simply a system form of method step [1c], q.v.

204. Therefore, Fukuda '98 discloses [9c].

5. **[9d] wherein the at least one distributed parallel computing program concurrently uses the at least one distributed sequential computing program and the at least one spawned child distributed sequential computing program to perform parallel processing and/or operations,**

205. Limitation [9d] is literally identical to limitation [1d], q.v.

206. Therefore, Fukuda '98 discloses limitation [9d].

6. **[9e] wherein the at least one distributed parallel computing program concurrently uses the at least one distributed sequential computing program and the at least one spawned child distributed sequential computing program to perform parallel processing and/or operations,**

207. Limitation [9e] is literally identical to limitation [1e], q.v.

208. Therefore, Fukuda '98 discloses limitation [9d] and discloses all of claim 9.

- J. **Claim 10: The system of claim 9, wherein the at least one distributed sequential computing program comprises at least one mobile agent.**

209. Fukuda '98 discloses claim 9, as discussed above.

210. Claim 7 recites “configuring the at least one distributed sequential computing program to include at least one mobile agent.” Hence see the discussion of claim 7, above.

211. Therefore, Fukuda '98 discloses claim 10.

K. Claim 11: The system of claim 10, wherein the at least one mobile agent is implemented using self-migrating threads.

212. Fukuda '98 discloses claim 10, as discussed above.

213. As discussed in connection with claims 2 and 8, the Messengers (i.e., mobile agents) are “autonomous messages,” “which carry their own behavior in the form of a program” (EX1008, 291) and are “capable of self-migration, i.e., autonomous navigation through both the logical and the physical networks.” *Id.*, 306.

214. Therefore, Fukuda '98 discloses claim 11.

L. Claim 12: The system of claim 10, wherein the at least one mobile agent is an explicit-navigation mobile agent.

215. Fukuda '98 discloses claim 10, as discussed above.

216. The '080 Patent explains explicit-navigation mobile agents:

In another aspect of navigational programming, an explicit-navigation mobile agent system may be built on top of a DSM. A DSM system may be used to serve as communication media for the globally accessible data, or in other words, agent variables.

EX1001, 10:18-22

217. “These self-migrating threads may be programmed explicitly, e.g., with ‘migration statements,’” wherein “[t]he source code to handle the explicit migration of threads may be in the form of the hop() command.” EX1001, 3:20-22;

3:29-30. The '080 Patent further states that “[s]elf-migrating threads may be in the form of mobile agents.” *Id.*, 8:21-22.

218. Fukuda '98 discloses explicit-navigation mobile agents by explaining that the source code that handles the explicit migration of threads is the *hop()* command: “**The *hop* Statement.** The *hop* statement permits a Messenger to move around the logical network and is based directly on the navigational calculus.” EX1008, 297; see also 298; 304-305 (Figure 4). Thus, the *hop* statement expresses explicit navigation.

219. Therefore, Fukuda '98 discloses claim 12.

M. Claim 13: The system of claim 10, wherein the at least one mobile agent is an implicit-navigation mobile agent.

220. Fukuda '98 discloses claim 10, as discussed above.

221. The '080 Patent explains what an implicit-navigation mobile agent is: “These self-migrating threads may be programmed . . . implicitly, e.g., driven by data distribution. With this approach, a thread, M, may perform its computations on one processor, Processor 1, and if the thread M needs to access memory belonging to another processor, e.g., Processor 2, it may suspend its computations, migrate, or move, to the other processor, Processor 2, and then resume its computations.” EX1001, 3:20-28.

222. Further, “In another aspect of navigational programming, an implicit-navigation mobile agent system may be based on a DSM. In this system, a

mechanism is included that decides for the agent when and where to migrate, and how the migration is achieved. FIG. 8 depicts a functional design of an implicit navigation agent system.” *Id.*, 10:36-41.

223. Fukuda '98 discloses implicit-navigation mobile agents: “In MESSENGERS, multiple destination nodes may be specified. The specification can be made implicitly, using the navigational calculus described in Section 2.3.” EX1008, 308.

224. Therefore, Fukuda '98 discloses claim 13.

N. **Claim 14: The system of claim 10, wherein the at least one mobile agent is configured to move from one processor to another processor.**

225. Fukuda '98 discloses claim 10, as discussed above.

226. As discussed in connection with claims 2 and 4, Fukuda '98 discloses that the distributed parallel program is configured to operate across multiple processors: “the individual entities move not only within the simulated space but actually migrate among processors in the network.” EX1008, 306. “This occurs when an entity crosses the boundary of a logical region. The corresponding Messenger performs a *hop* statement, which causes it to automatically migrate into the processor responsible for the corresponding logical region.” *Id.* Furthermore, the '080 patent discloses that a Messenger can be a mobile agent: “An example of such a system is MESSENGERS, in which applications are developed as

collections of mobile agents, which may be referred to as Messengers.” EX1001, 8:29-31.

227. Therefore, Fukuda '98 discloses claim 14.

O. Claim 15: The system of claim 10, wherein the at least one mobile agent operates within a distributed shared memory system.

228. Fukuda '98 discloses claim 10, as discussed above.

229. As described in Fukuda '98, MESSENGERS is a distributed shared memory system (at least because it uses “at least one distributed shared variable” per limitation 1[b]). Further, in MESSENGERS, “applications are developed as collections of mobile agents, which may be referred to as Messengers.” EX1001, 8:29-31.

230. Therefore, Fukuda '98 discloses claim 15.

P. Claim 16: The system of claim 9, wherein the at least one processor is located across a physical network.

231. Fukuda '98 discloses claim 9, as discussed above.

232. Fukuda '98 discloses that “[a] computation under MESSENGERS involves the dynamic construction of a logical network that is mapped onto the underlying physical network,” where the physical network is “the lowest level network (a LAN or WAN), which constitutes the underlying computational resource.” EX1008, 293, 294; see also Ex.1009 at 98-99.

233. Therefore, Fukuda '98 discloses claim 16.

Q. Claim 17: The method of claim 1, wherein the distributed parallel computing program is configured to operate across a distributed shared memory system.

234. Fukuda '98 discloses claim 1, as discussed above.

235. As described in Fukuda '98, MESSENGERS is a distributed shared memory system (at least because it uses “at least one distributed shared variable” per limitation 1[b]). MESSENGERS includes a distributed parallel computing program, as shown in connection with limitation 1[c].

236. Therefore, Fukuda '98 discloses claim 17.

R. Claim 18: The system of claim 10, wherein the at least one mobile agent is configured to move from one processor to another processor.

237. Fukuda '98 discloses claim 10, as discussed above.

238. The limitation of claim 18 is the same as the limitation of claim 17, q.v.

239. Therefore, Fukuda '98 discloses claim 18.

XI. CONCLUDING STATEMENT

240. Relying on my education, experience, skill, and training in the relevant field of art, and after reviewing the materials identified above, it is my opinion that all of the Challenged Claims are invalid as anticipated.

XII. JURAT

241. In signing this declaration, I recognize that the declaration will be filed as evidence in a contested case before the Patent Trial and Appeal Board of the

United States Patent and Trademark Office. I also recognize that I may be subject to cross-examination in the case and that cross-examination will take place within the United States. If cross-examination is required of me, I will appear for cross-examination remotely within the United States during the time allotted for such cross-examination at a time and place agreed by counsel.

242. I declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true, and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

Executed September 22, 2025 at Pittsburgh, PA.

By: Michael Ian Shamos
Michael Ian Shamos