IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

SERVICENOW, INCORPORATED,

Petitioner,

v.

INQUISIENT INCORPORATED

Patent Owner.

IPR2023-01071

U.S. Patent No. 7,979,468

DECLARATION OF JULES WHITE IN SUPPORT OF PETITION FOR *INTER PARTES* REVIEW

TABLE OF CONTENTS

Page

DECLARATION OF JULES WHITE IN SUPPORT OF 1
PETITION FOR *INTER PARTES* REVIEW 1
I. INTRODUCTION 1
II. EDUCATION AND WORK EXPERIENCE 1
III. COMPENSATION 5
IV. LEGAL PRINCIPLES 6
V. LEVEL OF SKILL IN THE ART 8
VI. OVERVIEW OF THE '468 PATENT 9
VII. PROSECUTION OF THE '468 PATENT 11
VIII. CLAIM CONSTRUCTION 12
IX. BACKGROUND TECHNOLOGY 12
A. Object Oriented Software Development Models 12
X. CITED ART 20
A. Althoff (Ex. 1006) 20
B. Surjanto (Ex. 1008) 21
1. Surjanto is a Prior Art Printed Publication 21
C. Nye (Ex. 1005) 24
D. Moore (Ex. 1007) 25
XI. GROUND 1: ALTHOFF IN VIEW OF NYE RENDERS OBVIOUS CLAIMS 1-9, 11-22, 24-35, 37-39 26
1. A POSITA Would Have Been Motivated To Combine Motivation to Combine Althoff and Nye 26
2. Claim 1 33
(a) 1[pre]: A computerized system for manipulating data sets, comprising: 33
(b) 1[a]: a processor; and 34
(c) 1[b]: a data repository, the data repository being adapted to process, retrieve, and store

data contained in the data sets and one or more layers of metadata of the data in the data sets, comprising: 35

(d) 1[c]: an element module configured to store and uniquely identify a plurality of elements, wherein each of the elements is identified by a unique element identification;..... 38

(e) 1[d]: an element relation module configured to store one or more relationships between the elements in the element module; 40

(f) 1[e]: a class module configured to define at least one class of the elements and store the class; 43

(g) 1[f]: an attribute module configured to define one or more attributes and store the attributes; 44

(h) 1[g]: a class attribute module configured to define and store one or more class-attribute associations between at least one of the attributes and the class;..... 46

(i) 1[h]: a type definition module configured to define and store one or more types of the class, the attributes related to the class, and the relationships between the elements; 49

(j) 1[i] a state machine module configured to store one or more state machine types associated with at least one of the elements; and 53

(k) 1[j] a status module configured to store one or more statuses of each state machine. 54

2. Claim 2: The computerized system of claim 1, wherein the data repository further comprises: a state machine transition module configured to store a transition from a first status to a second status. 56

3. Claim 3: The computerized system of claim 2, wherein the state machine module defines and stores at least one forward transition connection, the forward transition connection restricting the transition from the first status to the second status..... 57

4. Claim 4: The computerized system of claim 1, wherein the type definition module defines a type including at least one of a boolean type, a picklist type, and a state machine type. 60

5. Claim 5: The computerized system of claim 4, wherein the data repository further comprises: a list option module configured to enumerate one or more picklist attributes of the picklist type. 61

6. Claim 6: The computerized system of claim 2, wherein the data repository further comprises: a named text module configured to store textual representations of query, control, and display mechanisms that relate to the elements. 62

7. Claim 7: The computerized system of claim 6, wherein the data repository further comprises: a tuple module configured to generate and store one or more tuples, wherein each of the tuples associates a subset of the elements in accordance to one of the types associated with the subset of the elements. 64

8. Claim 8: The computerized system of claim 1, wherein each of the class-attribute associations is identified by a logical name and a name of the class. 67

9. Claim 9: The computerized system of claim 8, wherein the data repository further comprises: 68

   (a) an element attribute module configured to define and store the attributes of the elements in the class 68

   (b) wherein the element attribute module is identified using the name of the class. 69

10. Claim 11: The computerized system of claim 10, wherein the data repository further comprises: an element document module configured to store one or more formatting data for grouping and displaying a subset of the elements. 71

11. Claim 12: The computerized system of claim 1, wherein the processor is configured to generate a data model of the elements based on the data in the data sets and the layers of metadata stored in the data repository..... 71

12. Claim 13: The computerized system of claim 1, wherein the processor is configured to generate a dynamic user-interface for accessing and manipulating the elements based on the data in the data sets and the layers of metadata stored in the data repository..... 73

13. Claims 14-22, 24-35, 37-39 75

XII. GROUND 2: ALTHOFF IN VIEW OF NYE IN FURTHER VIEW OF SURJANTO RENDERS OBVIOUS CLAIMS 10, 23, 36 78

1. A POSITA Would Have Been Motivation to Combine Althoff, Nye, and Surjanto..... 78

2. Claim 10: The computerized system of claim 9, wherein the data repository further comprises: an element history module configured to store one or more indicators of modification to the elements. 83

3. Claims 18, 20, 23, 31, 33, 36 85

XIII. GROUND 3: ALTHOFF IN VIEW OF NYE IN FURTHER VIEW OF MOORE RENDERS OBVIOUS CLAIMS 6-7, 19-20, 32-33 85

1. Motivation to Combine Althoff, Nye, and Moore..... 85

2. Claim 6: The computerized system of claim 2, wherein the data repository further comprises: a named text module configured to store textual representations of query, control, and display mechanisms that relate to the elements. 91

3. Claim 7: The computerized system of claim 6, wherein the data repository further comprises: a tuple module configured to generate and store one or more tuples, wherein each of the tuples associates a subset of the elements in accordance to one of the types associated with the subset of the elements. 96

4. Claims 19-20, 32-33 99

XIV. SECONDARY CONSIDERATIONS 100

XV. CONCLUSION 100

LIST OF EXHIBITS

Ex.	Description
1001	U.S. Patent No. 7,979,468 ("'468 Patent")
1002	File History for U.S. Patent No. 7,979,468 ("File History")
1003	Declaration of Dr. Jules White ("White Declaration")
1004	Curriculum Vitae of Dr. Jules White
1005	U.S. Patent No. 6,341,279 ("Nye")
1006	U.S. Patent No. 6,374,252 ("Althoff")
1007	U.S. Patent No. 7,627,552 ("Moore")
1008	XML Content Management Based on Object-Relational Database Technology, Surjanto *et al.*, January 2000 ("Surjanto")
1009	Intentionally Left Blank
1010	Declaration of Dr. Sylvia Hall-Ellis, Library Information Science Professor ("Hall-Ellis Declaration")
1011	Joint Claim Construction Statement, Case No. 1:22-cv-00900-CJB (D. Del.), ECF No. 85 ("Claim Construction")
1012	Scheduling Order, Case No. 1:22-cv-00900-CJB (D. Del.), ECF No. 36 ("Scheduling Order")
1013	Transcript of Motion to Dismiss, Case No. 1:22-cv-00900-CJB (D. Del.), ("Section 101 Hearing")
1014	Complaint, Case No. 1:22-cv-00900-CJB (D. Del.), ECF No. 1 ("Complaint")
1015	Notice of Service, Case No. 1:22-cv-00900-CJB (D. Del.), ECF No. 6 ("Notice of Service")

LIST OF CHALLENGED CLAIMS

Claim	Limitation
1[pre]	A computerized system for manipulating data sets, comprising:
1[a]	a processor; and
1[b]	a data repository, the data repository being adapted to process, retrieve, and store data contained in the data sets and one or more layers of metadata of the data in the data sets, comprising:
1[c]	an element module configured to store and uniquely identify a plurality of elements, wherein each of the elements is identified by a unique element identification;
1[d]	an element relation module configured to store one or more relationships between the elements in the element module;
1[e]	a class module configured to define at least one class of the elements and store the class;
1[f]	an attribute module configured to define one or more attributes and store the attributes;
1[g]	a class attribute module configured to define and store one or more class-attribute associations between at least one of the attributes and the class;
1[h]	a type definition module configured to define and store one or more types of the class, the attributes related to the class, and the relationships between the elements;
1[i]	a state machine module configured to store one or more state machine types associated with at least one of the elements; and
1[j]	a status module configured to store one or more statuses of each state machine.

Claim	Limitation
2	The computerized system of claim 1, wherein the data repository further comprises: a state machine transition module configured to store a transition from a first status to a second status.
3	The computerized system of claim 2, wherein the state machine module defines and stores at least one forward transition connection, the forward transition connection restricting the transition from the first status to the second status.
4	The computerized system of claim 1, wherein the type definition module defines a type including at least one of a boolean type, a picklist type, and a state machine type.
5	The computerized system of claim 4, wherein the data repository further comprises: a list option module configured to enumerate one or more picklist attributes of the picklist type.
6	The computerized system of claim 2, wherein the data repository further comprises: a named text module configured to store textual representations of query, control, and display mechanisms that relate to the elements.
7	The computerized system of claim 6, wherein the data repository further comprises: a tuple module configured to generate and store one or more tuples, wherein each of the tuples associates a subset of the elements in accordance to one of the types associated with the subset of the elements.
8	The computerized system of claim 1, wherein each of the class-attribute associations is identified by a logical name and a name of the class.
9[pre]	The computerized system of claim 8, wherein the data repository further comprises:
9[a]	an element attribute module for each of the at least one class, the element attribute module configured to define and store the attributes of the elements in the class,

Claim	Limitation
9[b]	wherein the element attribute module is identified using the name of the class.
10	The computerized system of claim 9, wherein the data repository further comprises: an element history module configured to store one or more indicators of modification to the elements.
11	The computerized system of claim 10, wherein the data repository further comprises: an element document module configured to store one or more formatting data for grouping and displaying a subset of the elements.
12	The computerized system of claim 1, wherein the processor is configured to generate a data model of the elements based on the data in the data sets and the layers of metadata stored in the data repository.
13	The computerized system of claim 1, wherein the processor is configured to generate a dynamic user-interface for accessing and manipulating the elements based on the data in the data sets and the layers of metadata stored in the data repository.
14[pre]	A method for manipulating data sets, comprising:
14[a]	storing and uniquely identifying a plurality of elements in a data repository, each of the elements identified by a unique element identification, wherein the data repository is adapted to process, retrieve, and store data contained in the data sets and one or more layers of metadata of the data in the data sets;
14[b]	storing, in the data repository, one or more relationships between the elements;
14[c]	defining at least one class of the elements and storing the class in the data repository;

Claim	Limitation
14[d]	defining one or more attributes and storing the attributes in the data repository;
14[e]	defining one or more class-attribute associations between at least one of the attributes and the class and storing the class-attribute associations in the data repository;
14[f]	defining one or more types of the class, the attributes related to the class, and the relationships between the elements, and storing the types in the data repository;
14[g]	defining one or more state machines and defining and storing a type for each state machine associated with at least one of the elements; and
14[h]	defining one or more statuses for each state machine and storing the statuses in the data repository.
15	The method of claim 14, further comprising: defining and storing a transition from a first status to a second status in the data repository.
16	The method of claim 15, wherein the type for each state machine comprises at least one forward transition connection, the forward transition connection restricting the transition from the first status to the second status.
17	The method of claim 14, wherein the one or more types of the class comprise a type including at least one of a boolean type, a picklist type, and a state machine type.
18	The method of claim 17, further comprising: enumerating one or more picklist attributes of the picklist type and storing the enumerated picklist attributes in the data repository.
19	The method of claim 15, further comprising: storing, in the data repository, textual representations of query, control, and display mechanisms that relate to the elements.

Claim	Limitation
20	The method of claim 19, further comprising: generating one or more tuples, each of the tuples associating a subset of the elements in accordance to one of the types associated with the subset of the elements, and storing the tuples in the data repository.
21	The method of claim 14, wherein each of the class-attribute associations is identified by a logical name and a name of the class.
22	The method of claim 21, further comprising: defining, for each of the at least one class, a set of the attributes of the elements in the class, wherein the set is identified using the name of the class, and storing the set in the data repository.
23	The method of claim 22, further comprising: storing, in the data repository, one or more indicators of modification to the elements.
24	The method of claim 23, further comprising: storing, in the data repository, one or more formatting data for grouping and displaying a subset of the elements.
25	The method of claim 14, further comprising: generating a data model of the elements based on the data in the data sets and the layers of metadata stored in the data repository.
26	The method of claim 14, further comprising: generating a dynamic user-interface for accessing and manipulating the elements based on the data in the data sets and the layers of metadata stored in the data repository.
27[pre]	A non-transitory computer-readable medium storing a computer-executable program which, when executed by a processor performs a method comprising:

Claim	Limitation
27[a]	storing and uniquely identifying a plurality of elements in a data repository, each of the elements identified by a unique element identification, wherein the data repository is adapted to process, retrieve, and store data contained in the data sets and one or more layers of metadata of the data in the data sets;
27[b]	storing, in the data repository, one or more relationships between the elements;
27[c]	defining at least one class of the elements and storing the class in the data repository;
27[d]	defining one or more attributes and storing the attributes in the data repository;
27[e]	defining one or more class-attribute associations between at least one of the attributes and the class and storing the class-attribute associations in the data repository;
27[f]	defining one or more types of the class, the attributes related to the class, and the relationships between the elements, and storing the types in the data repository;
27[g]	defining one or more state machines and defining and storing a type for each state machine associated with at least one of the elements; and
27[h]	defining one or more statuses for each state machine and storing the statuses in the data repository.
28	The computer-readable medium of claim 27, wherein the method further comprises: defining and storing a transition from a first status to a second status in the data repository.
29	The computer-readable medium of claim 28, wherein the type for each state machine comprises at least one forward transition connection, the forward transition connection restricting the transition from the first status to the second status.

Claim	Limitation
30	The computer-readable medium of claim 27, wherein the one or more types of the class comprise a type including at least one of a boolean type, a picklist type, and a state machine type.
31	The computer-readable medium of claim 30, wherein the method further comprises: enumerating one or more picklist attributes of the picklist type and storing the enumerated picklist attributes in the data repository.
32	The computer-readable medium of claim 28, wherein the method further comprises: storing, in the data repository, textual representations of query, control, and display mechanisms that relate to the elements.
33	The computer-readable medium of claim 32, wherein the method further comprises: generating one or more tuples, each of the tuples associating a subset of the elements in accordance to one of the types associated with the subset of the elements, and storing the tuples in the data repository.
34	The computer-readable medium of claim 27, wherein each of the class-attribute associations is identified by a logical name and a name of the class.
35	The computer-readable medium of claim 34, wherein the method further comprises: defining, for each of the at least one class, a set of the attributes of the elements in the class, wherein the set is identified using the name of the class, and storing the set in the data repository.
36	The computer-readable medium of claim 35, wherein the method further comprises: storing, in the data repository, one or more indicators of modification to the elements.
37	The computer-readable medium of claim 36, wherein the method further comprises: storing, in the data repository, one or more formatting data for grouping and displaying a subset of the elements.

Claim	Limitation
38	The computer-readable medium of claim 27, wherein the method further comprises: generating a data model of the elements based on the data in the data sets and the layers of metadata stored in the data repository.
39	The computer-readable medium of claim 27, wherein the method further comprises: generating a dynamic user-interface for accessing and manipulating the elements based on the data in the data sets and the layers of metadata stored in the data repository.

GROUNDS OF CHALLENGE (37 C.F.R. § 42.204(b)(2))

Ground	Reference(s)	Basis	Claims
1	Althoff and Nye	§ 103	1-9, 11-22, 24-35, 37-39
2	Althoff, Nye, Surjanto	§ 103	10, 23, 36
3	Althoff, Nye, Moore	§ 103	6, 7, 19-20, 32-33

I. INTRODUCTION

1. I have been retained on behalf of ServiceNow, Inc. ("Petitioner") in connection with the above-captioned *inter partes* review (IPR). I have been retained to provide my opinions in support of Petitioner's Petition for *inter partes* review of U.S. Patent No. 7,979,468 (the "'468 patent").

2. My opinions are based on information currently available to me. To the extent that additional information becomes available, I reserve the right to continue my investigation and study, which may include a review of documents and information that recently have been or may be produced, as well as testimony from depositions that may yet be taken in this case. I may thus expand or modify my opinions as my investigation and study continues. I may also supplement my opinions in response to any additional information that becomes available to me or that Patent Owner makes available, any matters raised by Patent Owner and/or opinions provided by Patent Owner's experts, or in light of any relevant orders from the Patent Trial and Appeal Board.

II. EDUCATION AND WORK EXPERIENCE

3. A copy of my curriculum vitae that fully sets forth my qualifications can be found at Ex. 1004. Below is a summary of my education, work experience, and other qualifications.

4. I am currently the Associate Dean for Strategic Learning Programs and an Associate Professor of Computer Science and Computer Engineering at Vanderbilt University.

5. I have over 20 years of experience in computer science and computer engineering. I received a Bachelor's degree in Computer Science from Brown University, and a Bachelor's of Science and Ph.D. in Computer Science from Vanderbilt University. I am currently an Associate Professor in the Department of Computer Science at Vanderbilt University. I am also Associate Dean of Strategic Learning Programs in the School of Engineering. I was also previously an Assistant Professor in Computer Engineering at Virginia Tech.

6. I have held a variety of positions within industry and research affiliations with federally-supported research labs. Proceeding in reverse chronological order: From 2014-2020, I was President of Ziiio, Inc, a company that I co-founded. From 2012-2015, I was Chief Technology Officer of Optio Labs, Inc., a company that I co-founded. From 2012-2016, I was the Chief Scientist of Cloudpoint Labs, a company that I co-founded. From 2011-2012, I was a Visiting Research Scientist at the Carnegie Mellon Software Engineering Institute, a federally-funded research and development center. From 2001-2002, I worked as a software engineer at IBM.

7. I have published over 160 papers on topics including model-driven engineering, cyber-security, and cloud computing. According to Google Scholar, there are 6,500 citations to these papers. In addition to publishing on a variety of research topics, I have received several important distinctions. I received a National Science Foundation (NSF) CAREER Award, which is one of the NSF's "most prestigious awards in support of early-career faculty who have the potential to serve as academic role models in research and education and to lead advances in the mission of their department or organization." I have also received several Best Paper Awards for research work in domains ranging from Software Product-lines to Civil Engineering to Computer Vision.

8. As a faculty member, I run the Mobile Application computinG, optimizatioN, and secUrity Methods (MAGNUM) Group at Vanderbilt University. As a Principal Investigator or Co-Principal Investigator, I have received over $17,000,000 in grant support, both from federal entities, such as the National Science Foundation (NSF), Defense Advanced Research Projects Agency (DARPA), Air Force Research Labs (AFRL), and Office of Naval Research (ONR), and private entities, such as Varian Medical Systems and NOAH Basketball. My research focuses on securing, optimizing, and leveraging data from mobile cyber-physical systems. My mobile cyber-

physical systems ("CPS") research spans four key focus areas: (1) mobile, cloud, and CPS security and data collection, (2) high-precision mobile understanding of the world through indoor positioning and augmented reality, (3) CPS, mobile device and cloud infrastructure power and configuration optimization, and (4) applications of mobile cyber-physical systems in multi-disciplinary domains, including manufacturing, energy-optimized cloud computing, smart grid systems, healthcare/manufacturing security, next-generation construction technologies, citizen science, and augmented reality.

9. My research work has been transitioned to industry where it has both won patent protection and led to venture-backed startup companies. My work on securing mobile devices was licensed into Optio Labs, a company that I co-founded. Optio Labs received over $10,000,000 in venture capital backing. My work on mobile augmented reality was licensed into PAR Works / Cloudpoint Labs, a company that I co-founded. The company received $1,000,000 in venture backing and won awards, such as being an Innovation Award Honoree at CES and a finalist for Technical Achievement Award at SXSW. I am an inventor on multiple patents.

10. My research and teaching have dealt extensively with cloud computing. My startup company, PAR Works / Cloudpoint Labs, developed mobile augmented reality systems that used high-performance servers in the

cloud to augment the limited computing capabilities of mobile devices. A key component of this work was handling rapid scaling of cloud virtual machines to meet tight latency requirements of mobile augmented reality applications. Optio Labs, which I co-founded and worked as the CTO for, built cloud-based systems for managing and controlling security on mobile devices. In my research work, I have investigated optimization algorithms for resource allocation and autoscaling in the cloud. In addition, I have taught various classes on cloud computing as part of Vanderbilt's partnership with Coursera, such as "Building Cloud Services with the Java Spring Framework," that have had over 70,000 students enrolled.

11. In addition to the summary, I provide in this Report, I describe my education, training, research, and experience in greater detail in my CV, attached as Ex. 1012.

III. COMPENSATION

12. I am being compensated for my time at the rate of $500 per hour. I have no interest in the outcome of this proceeding. My compensation does not depend in any way upon the outcome of this proceeding, the opinions I express, or the content of my testimony.

IV. LEGAL PRINCIPLES

13. I am informed that "prior art" includes patents and printed publications that existed before the earliest applicable filing date of the '468 Patent.

14. I am informed that in order for a claim to be anticipated, each and every requirement of the claim must be found, expressly or inherently, in a single prior art reference as recited in the claim.

15. I am informed that a claimed invention is not patentable if the claimed invention would have been obvious to a person of ordinary skill in the field of the invention at the time the invention was made.

16. I am informed that in order to show obviousness based on a combination of references, a particular motivation to combine the teachings in the references must be shown. I am informed that in order to show obviousness based on a combination of references, a particular motivation to combine the teachings in the references must be shown. I may consider whether the change was merely the predictable result of using prior art elements according to their known functions, or whether it was the result of true inventiveness. I may also consider whether there is some teaching or suggestion in the prior art to make the modification or combination of elements claimed in the patent. I may consider whether the innovation applies

a known technique that had been used to improve a similar device or method in a similar way. I may also consider whether the claimed invention would have been obvious to try, meaning that the claimed innovation was one of a relatively small number of possible approaches to the problem with a reasonable expectation of success by those skilled in the art.

17. I am informed that the obviousness inquiry should not be done in hindsight, and depends on the scope and content of the prior art, the differences between the prior art and the claims at issue, the knowledge of a person of ordinary skill in the pertinent art at the time of invention, and any other objective factors indicating obviousness or non-obviousness.

18. I am informed that in order to rely on a reference for obviousness, the reference must be analogous art. I also understand that to be analogous art, the reference must be either (1) from the same field of endeavor as the claimed subject matter, regardless of the problem addressed, or (2) if not in the same field of endeavor, reasonably pertinent to the particular problem with which the inventor is involved. I am also familiar with the premise that for a reference to be reasonably pertinent, it must have logically commended itself to an inventor's attention at the time of invention.

19. I am informed that claim terms are generally given their ordinary and customary meaning, which is the meaning that the term would have to a

person of ordinary skill in the art at the time of the invention. I further understand that a person of ordinary skill in the art must read the claim term not only in the context of the particular claim in which the term appears but in the context of the entire patent, including the specification.

20. I understand that in this *inter partes review* the claims must be given interpreted under the *Phillips* standard, that is, in accordance with its plain and ordinary meaning as understood by a person of ordinary skill in the art.

V. LEVEL OF SKILL IN THE ART

21. I understand that a person of ordinary skill in the relevant art (also referred to herein as "person of ordinary skill in the art" or "POSITA") is presumed to be aware of all pertinent art, thinks along conventional wisdom in the art, and is a person of ordinary creativity—not an automaton.

22. I have been asked to consider the level of ordinary skill in the field that someone would have had at the time the claimed invention was made. In deciding the level of ordinary skill, I considered the following:

- the levels of education and experience of persons working in the field;
- the types of problems encountered in the field; and
- the sophistication of the technology.

23. A person of ordinary skill in the art at the time of the alleged invention ("POSITA") would have a Bachelor's degree in Computer Science and three to five years of industry experience in application development, database development, enterprise application management; or a Ph.D. specializing in the field. This level of skill is approximate, and more experience would compensate for less experience and vice versa.

24. I am well-qualified to determine the level of ordinary skill in the art and am personally familiar with the technology of the '468 patent. I was a person of at least ordinary skill in the art at the time of the priority date of the '468 patent in 2005. Regardless, if I do not explicitly state that my statements below are based on this timeframe, all of my statements are to be understood as a person of ordinary skill in the art would have understood something as of the priority date of the '468 patent.

VI. OVERVIEW OF THE '468 PATENT

25. U.S. Pat. No. 7,979,468 (the "'468 patent") was filed on Jun. 14, 2006 and issued on July 12, 2011. The '468 patent claims priority to a provisional application filed on June 14, 2005.

26. The '468 patent claims that conventional "data storage systems" provide their own data model, which "lack[s] flexibility and portability." Ex. 1001, 1:35-36. Thus, the patent purports to provide "systems and methods

that store, manage, index, interrelate, and/or retrieve data sets in a manner independent of the data model." *Id.*, 1:50-53.

27. In actuality, what the '468 patent discloses is a number of database tables that allow for object-oriented ("OO") data modeling, which stores object types called classes, class attributes, and relationships between classes (specifically inheritance). These tables are shown in, for example, Figures 1 and 2. For example, Figure 2 contains a "class table 150," which stores classes, and just like how object classes contain attributes, "[a] class_attribute table 112 may associate a given attribute to one or more classes (in class table 150)." *Id.*, 5:6-7. Multiple classes can have the same attribute (a well-known feature of class inheritance) using the "class_attribute table." *Id.*, 5:7-9.

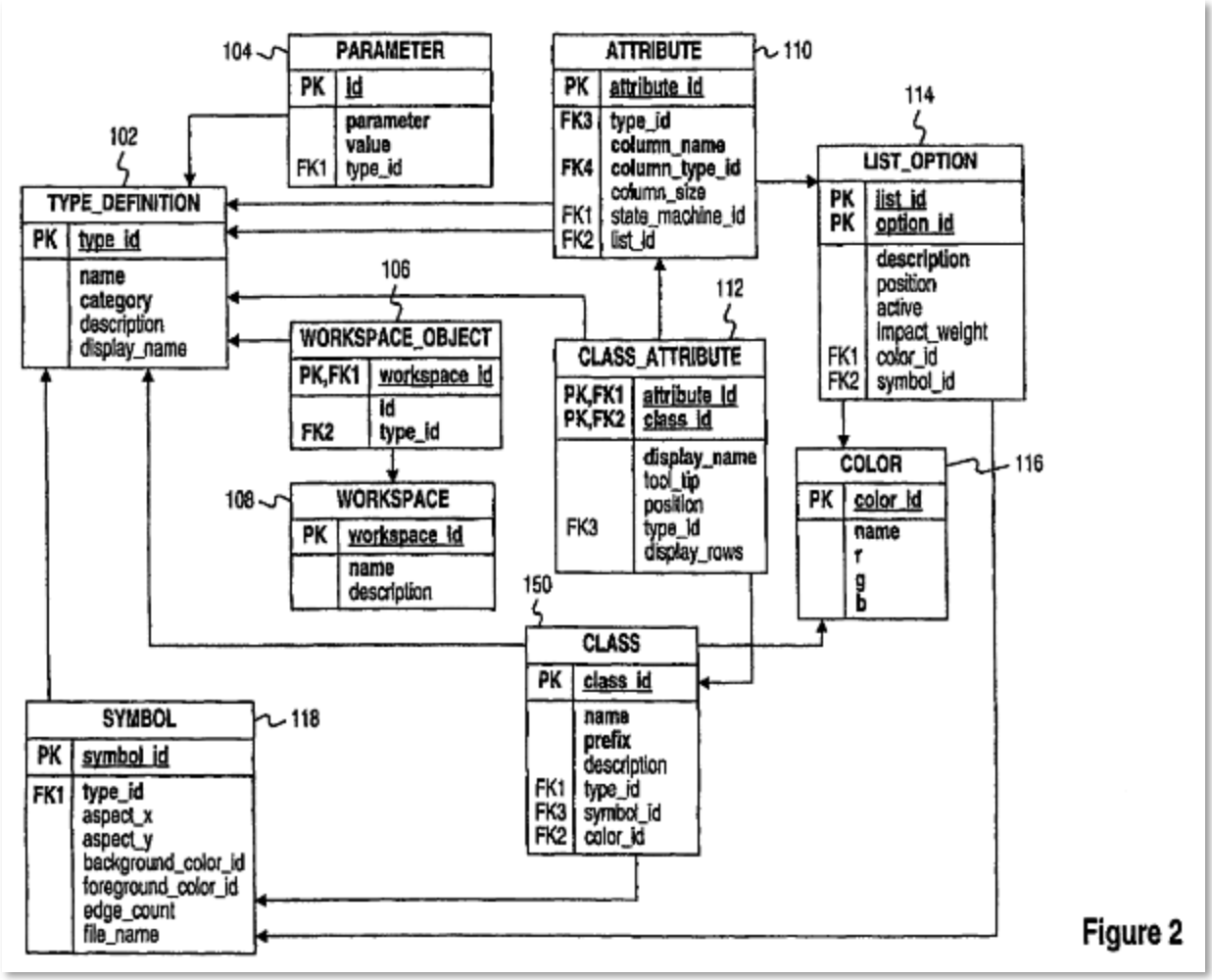


Figure 2

Id., Fig. 2.

VII. PROSECUTION OF THE ’468 PATENT

28. The ’468 patent was filed on June 3, 2011 as U.S. Patent Application No. US11/917,410. The initial set of claims were largely the same as issued, except that the independent claims did not recite the “state machine module configured to store one or more state machine types associated with at least one of the elements” or the “a status module configured to store one or more statuses of each state machine.”

29. The claims were ultimately amended to import limitations from the dependent claims, including the "state machine module" and the "status module" Ex. 1002, Examiner Amendments (April 12, 2011); *Id.*, Examiner Interview Summary Record. The claims were thereafter allowed. *Id.*, Notice of Allowance (April 12, 2011).

VIII. CLAIM CONSTRUCTION

30. It is my understanding that the patent owner submits that express interpretations of the challenged claims are not required to resolve this petition. Ex. 1011. I currently see no reason to disagree with this position, however, I reserve the right to modify my opinions in light of to any claim constructions established by the Board or any Court in the future.

31. I understand counsel for patent owner stated in a hearing earlier in the case that a 'module' refers to a "table or portion of a table." Ex. 1013, 93:17-19. This is consistent with how a POSITA would understand a 'module' and is a definition I have applied in my analysis herein."

IX. BACKGROUND TECHNOLOGY

A. Object Oriented Software Development Models

32. The tools and approaches for developing software systems have evolved over time to move from systems that require designers to use low-level abstractions to higher-level abstractions that are closer to the domain being described. For example, a very low level of abstraction is the low-level

machine code that describes operations on the processor of the computer that requires describing software in terms of movement of data between different memory addresses and applying low-level operations, such as "move the data in this memory address to this other memory address." Building complex software systems using these low-level operations, however, is difficult because there is a large semantic gap between operations like "moving data from one memory address to another" and a real-world domain, such as the software to control the engine of a car. It is difficult for domain experts, such as automotive engineers, to see how the software concepts map to the real-world automotive requirements that need to be met.

33. Over time, software engineering has evolved to use higher levels of abstraction, such as OO programming languages, that allow developers to describe solutions in terms of "classes" and "objects" that describe the domain. These higher level abstractions specify the software in a way that it more directly maps to the requirements for the domain. For example, software for the automotive domain might have a "Car" class with operations for "startEngine" and "stopEngine." An automotive domain expert can then more easily understand how the high-level "Car" class and its "startEngine" operation map to automotive requirements that the expert cares about.

34. OO programming languages were one of the major innovations that emerged in the 1980s[1]. OO programming languages describe software in terms of classes, attributes, objects, types, relationships, states, hierarchies, and operations. A *class* is an abstraction that can be roughly thought of as a "type" that describes a set of data and operations related to that data. For example, a "Car" could be a class that describes the concept of a car and the attributes and operations common to a car, such as its "color" attribute and operations for "startEngine" and "stopEngine".

35. An *object* is an instance of a class. For example, there is an abstract concept of a car, but there are many different instances of cars. For example, my car is one instance and my wife's car is another instance. In the OO programming realm, my car and my wife's car would each be a separate object with their own values for the color attribute. Similarly, invoking the "startEngine" operation on my car's object will start the engine for my car while calling the "startEngine" operation on her car object will start the engine in her car. Each object in an OO program has a unique identifier, which may take many forms, including its memory address.

[1] Stroustrup, Bjarne. "What is object-oriented programming?." *IEEE software* 5, no. 3 (1988): 10-20.

36. OO programs are built on relationships between classes and objects. For example, *inheritance* is a relationship between a child class and a parent class that indicates a form of specialization. A "Toyota4Runner" class can inherit from a "Car" class to indicate that it is a specialized type of car. If the "Car" class has a "color" attribute, then the "Toyota4Runner" class will inherit the "color" attribute and also have it. The "Toyota4Runner" can then specialize by adding additional attributes, operations, etc. that are not common to all cars, such as an attribute for "hasToyotaNavigationSystem", which would be a specialized attribute of Toyota 4Runner that isn't common to all cars. Inheritance an define a hierarchy of classes.

37. Another type of relationship in OO programs is *composition*. A "Car" class can have a relationship with "Engine" and "Door" classes. A car is composed of an engine and 2-4 doors. Every "Car" object will have an instance of an "Engine" object and 2-4 "Door" objects. Both classes and objects can participate in relationships. Composition is one way of defining a hierarchy of objects, such as a car contains an engine and an engine contains spark plugs.

38. Since at least the early 1990s, state machines have been used hand in hand with OO programs, both to describe object oriented systems and

to implement their underlying logic[2]. State machines define a series of "states" and "transitions" between the states. For example, a "Car" object could have an "engineStarted" state. A car object could transition from the "off" state to the "engineStarted" state when the key is turned in the ignition. States may also have "guard", "pre", or "post" conditions that restrict a transition, such as a guard that prevents transitioning from the "off" to the "engineStarted" state if the key is not in the ignition.

39. One of the innovations in software engineering has been techniques for mapping high-level abstractions down to low-level abstractions. An OO programming language has a "compiler", which is a computer program that translates OO classes, attributes, relationships, etc. into low-level instructions, such as "move the data at this memory address to this other memory address." The compiler alleviates the programmer from having to worry about how to perform this complex translation and performs it on the developer's behalf.

40. The success of OO programming led to its use in a variety of different types of systems, including databases. For example, in the early

[2] Sane, Aamod, and Roy Campbell. "Object-oriented state machines: Subclassing, composition, delegation, and genericity." *ACM Sigplan Notices* 30, no. 10 (1995): 17-32.

1990s, OO relational databases were developed[3]. These systems allowed developers to directly store, classes, objects, attributes, relationships, states, etc. in the underlying databases[4]. Moreover, they supported operations on them, such as stored procedures[5], which are named queries. OO programming can also work with object-relational mapping systems that translate classes, relationships, attributes, objects, etc. into database tables if the database does not directly support object storage[6]. These object-relational mapping systems work similar to a compiler, they translate classes, objects, etc. into database tables so that developers don't have to worry about determining how to perform this mapping themselves. These approaches can also directly store the representation of the classes, relationships, attributes, etc. in the database as tables, which are often named "class" table or "attribute" table as was done by Hora et al. in 1993[7].

---

[3] Premerlani, William J., James E. Rumbaugh, Michael R. Blaha, and Thomas A. Varwig. "An object-oriented relational database." *Communications of the ACM* 33, no. 11 (1990): 99-109.

[4] Sutherland, Jeff, Matthew Pope, and Ken Rugg. "The Hybrid Object-Relational Architecture (HORA) an integration of object-oriented and relational technology." In *Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing: states of the art and practice*, pp. 326-333. 1993.

[5] Eisenberg, Andrew. "New standard for stored procedures in SQL." *ACM SIGMOD Record* 25, no. 4 (1996): 81-88.

[6] Sutherland, Jeff, Matthew Pope, and Ken Rugg. "The Hybrid Object-Relational Architecture (HORA) an integration of object-oriented and relational technology." In *Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing: states of the art and practice*, pp. 326-333. 1993.

[7] Sutherland, Jeff, Matthew Pope, and Ken Rugg. "The Hybrid Object-Relational Architecture (HORA) an integration of object-oriented and relational technology." In *Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing: states of the art and practice*, pp. 326-333. 1993.

41. One of the natural evolutions of OO programming was the realization that the important aspect was the model of the classes, attributes, relationships, etc. and that a software program could automatically translate that model into a variety of different implementation details, ranging from low-level programming instructions to database tables. The important aspect is to describe the model of the system in terms of classes, states, attributes, etc. and then a "model interpreter" (e.g., compiler) could turn the system description into the actual software by translating it into the lower-level machine operations, database tables, etc.

42. Model-driven engineering[8], which is also referred to as "model-driven development", is an approach to building software and systems that is built on this realization that the model describing the system is the important activity and that the low-level implementation of the system can be generated from this model. One of the most widely used languages for modeling systems is called the Unified Modeling Language[9] (UML), which emerged in the 1990s as a standard.

---

[8] Schmidt, D.C., 2006. Model-driven engineering. *Computer-IEEE Computer Society-*, *39*(2), p.25.

[9] Booch, Grady, Ivar Jacobson, and James Rumbaugh. "The unified modeling language." *Unix Review* 14, no. 13 (1996): 5.

43. UML describes systems using classes, attributes, relationships, objects, states, state transitions, hierarchies, and many other modeling constructs. These UML models are then translated into software, including database tables, rows, columns, etc. A large variety of tools have been developed to support model-driven engineering, including tools such as the Generic Modeling Environment[10], IBM's Rational Rose[11], and the Eclipse Modeling Framework[12]. All of these tools rely either directly on UML or variations of class, attribute, relationship, state, hierarchy, object, etc. modeling.

44. One of the innovations in model-driven engineering was the realization that a system could be changed dynamically by simply changing the model to adapt the system[13]. This type of dynamic change could be done after the system was already deployed and running without actually taking the system offline. An end-user, developer, or other domain expert could go and describe a new class, object, attribute, etc. and then the interpreter could

---

[10] Ledeczi, Akos, Miklos Maroti, Arpad Bakay, Gabor Karsai, Jason Garrett, Charles Thomason, Greg Nordstrom, Jonathan Sprinkle, and Peter Volgyesi. "The generic modeling environment." In *Workshop on Intelligent Signal Processing, Budapest, Hungary*, vol. 17, no. 01, p. 2001. 2001.

[11] Quatrani, Terry. *Visual modeling with Rational Rose 2000 and UML*. Addison-Wesley Professional, 2000.

[13] Oreizy, Peyman, Nenad Medvidovic, and Richard N. Taylor. "Architecture-based runtime software evolution." In *Proceedings of the 20th international conference on Software engineering*, pp. 177-186. IEEE, 1998.

dynamically update the system to implement the new aspect of the system. For example, a product manager for an automotive company might create a new class describing a new car about to be released and the unique attributes it has and relationships to new parts. The new car may also have specialized states, such as "autonomousDrivingOn" and "autonomousDrivingOff" that are associated with new features in the car.

X. CITED ART

A. Althoff (Ex. 1006)

45. U.S. Patent No. 6,374,252 to Althoff *et al.* was filed on October 16, 1997 and issued on April 16, 2002.

46. Althoff discloses that its invention is directed towards a database system and methods that "obtains the advantages of object-oriented database applications, while retaining the ability to use present relational database management systems." Ex. 1006, 3:6-9. Althoff further notes that it "would be advantageous to obtain the advantages of OO database management, while retaining the efficiency of, and the installed investment in, relational database management systems." *Id.*, 2:5-8. Althoff discloses that the "advantages of object-oriented" databases "including dynamically modifying the object hierarchy and object relationships, populating the database, dynamically populating and editing the objects in the database, and dynamically generating

and performing searches against the database." *Id.*, 2:28-33. Althoff discloses "[a]t least two types of relationship are modeled by the system, inheritance relationships and data-model relationships." *Id.*, 3:61-63.

B. Surjanto (Ex. 1008)

47. "XML Content Management Based on Object-Relational Database Technology" by Surjanto *et al.* was published by August 2002.

48. Surjanto discloses an "object-relational database management system" that "improves content management of XML documents" by "exploiting their structural information." Ex. 1008, 9. In particular, Surjanto discloses that "[w]ith the advent of object-relational database management systems (ORDBMSs) [Sto98], developers can extend the database system." *Id.*, 2. Surjanto "develop[ed] XCoP as part of an integrated Web database framework ... which is built by our database research group using the extensibility features of an ORDBMS." *Id.* By doing this, Surjanto provides benefits such as content reuse, exploitation of structural information, content consistency enforcement, etc.. Id., 71.

1. Surjanto is a Prior Art Printed Publication

49. Surjanto is a printed publication that qualifies as prior art.

50. I am familiar with the WISE conference and its publication practices, and the papers presented at the WISE conference would have been

published in its proceedings and readily available as of the publication of those proceedings. Moreover, Surjanto bears a 2000 date on the first page of the paper, further indicating its reliability. Ex. 1008, 9.

51. Surjanto was published in the IEEE's Proceedings of the First International Conference on Web Information Systems Engineering, following its presentation at a conference that occurred in Hong Kong from June 19 to 21, 2000. As is typical with papers presented at IEEE conferences, it was made available to the public when it was published in the physical proceedings (i.e., a book) of that conference in the same year, as reflected in the copyright date on the face of the paper. The paper is available on the IEEE Xplore website at https://ieeexplore.ieee.org/abstract/document/882377, and according to the website, has been available on that website since August 6, 2002, which in my experience and that of other POSITAs is a highly-reliable date indicative of its publication status. The IEEE-published Surjanto paper is attached as Exhibit 1008, and is identical to the paper retrieved by Dr. Hall-Ellis from library records. That Surjanto was publicly available in 2000 comports with my personal experience with the standard publication procedure for IEEE proceedings. Materials from IEEE conferences are, by standard procedure, published in a physical book in the days and weeks following the symposium.

52. Surjanto was presented in an IEEE conference and published by the IEEE. The Surjanto conference proceedings bears a publication date of 2000 which is inherently reliable and trustworthy given the publication source, the IEEE. Ex. 1010. Indeed, I understand that the Board has stated that the IEEE is a well-known, reputable compiler and publisher of scientific and technical publications, and that POSITAs rely on the copyright date provided in IEEE publications. I agree with the Board.

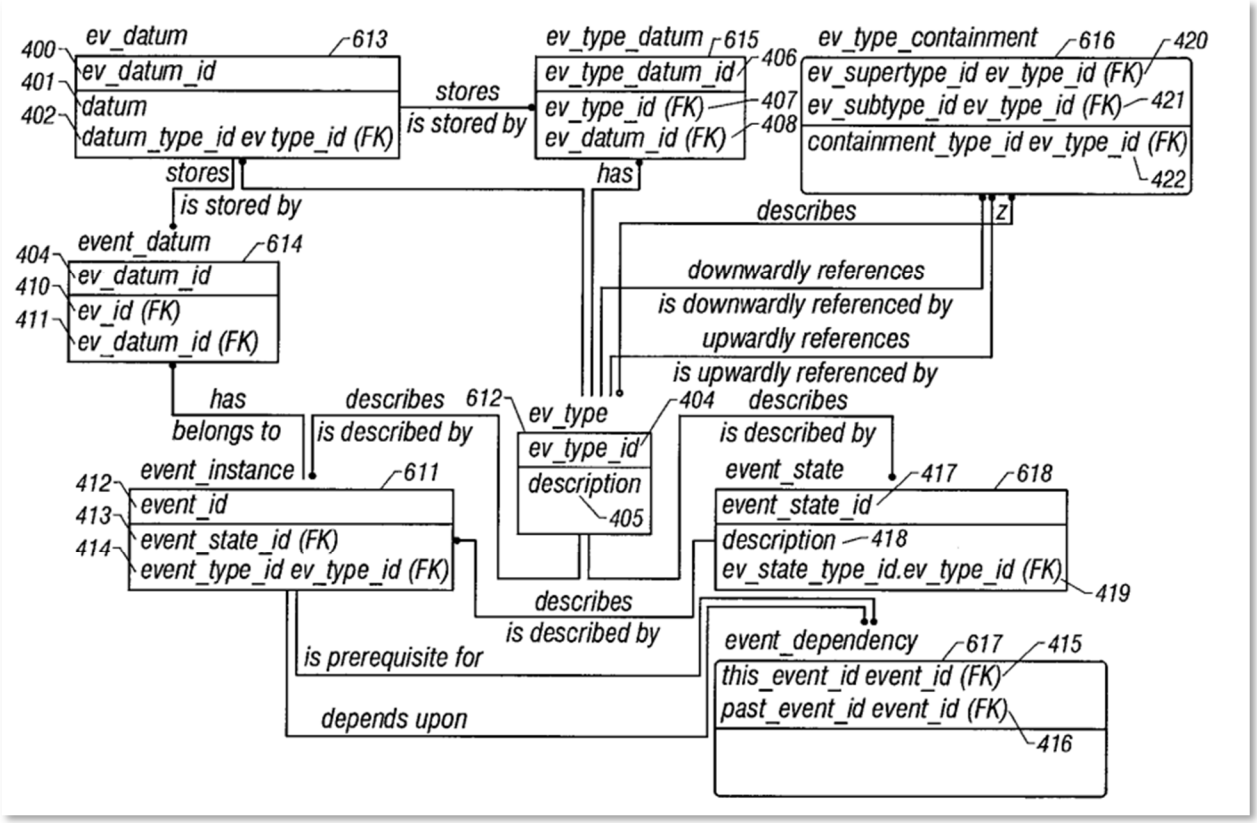
53. Moreover, the copies of Surjanto in the record are authentic. Exhibit 1008 is a complete copy of Surjanto that, according to Dr. Hall-Ellis, was obtained from the online catalog of the Linda Hall library and was available by 2000. Ex. 1010, ¶39. Based on my significant experience as a reader and author of IEEE publications, Exhibit 1008 is complete and has no visible alterations. Exhibit 1008 is identical to the copy available on IEEE Xplore, which was available by 2002. The IEEE Xplore database is exactly the place an IEEE-published paper would appear. Exhibit 1008 also has all the hallmarks of an authentic copy of an IEEE-published paper, including a cover page and copyright page indicating its publication in WISE, as well as a Table of Contents for those proceedings. Ex. 1008, 1-8. Thus, in my opinion, Exhibit 1008 is a true and authentic copy of Surjanto as it was published in 2000.

C. Nye (Ex. 1005)

54. U.S. Patent No. 6,341,279 to Nye was filed on October 12, 1998 and issued on January 22, 2002.

55. Nye discloses an event model "which allows a developer to add complex dependency logic to an existing database without having to modify the underlying structure of the database." Ex. 1005, 5:34-37 Nye discloses that the event model "contains one or more events." *Id.*, 10:58-61. Such event model and events are used to "represent[] a set of actions that are optionally contingent upon a condition." *Id.*, 10:62-63. Nye further discloses that events "each have an event state" which "is defined by the developer and varies according to the event that is created." *Id.*, 12:39-41.

56. Nye discloses a system for implementing event models, events, and event states that "can be instantiated to represent a number of different event models and the corresponding dependencies that interrelate them." *Id.*, 13:34-37. Figure 4, reproduced below, depicts the system for the event model, events, and event states.



Id., Fig. 4.

D. Moore (Ex. 1007)

57. U.S. Patent No. 7,627,552 to Moore *et al.* was filed on March 27, 2003 and issued on December 1, 2009.

58. Moore teaches a system that "is able to take a property that is stored in the database and represent it as a container that is like a folder." Ex. 1007, 3:39-41. Moore provides this using a virtual folder system, which "expose[s] regular files and folders (also known as directories) to users in different views based on their metadata instead of the actual physical underlying file system structure on the disk." *Id.*, 5:51-54. This "virtual

folder system 200" comprises a number of components, most notably a "folder processor 210, a relational database 230, [and] a virtual folder descriptions database 232." *Id.*, 8:5-9. One component of the system is a "query builder component 220 [that] builds SQL queries" from "data from the virtual folder descriptions database 232, including a list of types to display in the folder, the initial filter, and the physical location to show results from (the scopes)." *Id.*, 9:19-25. The "query builder component 220 sends the SQL queries to the relational database 230." *Id.*, 9:25-27.

XI. GROUND 1: ALTHOFF IN VIEW OF NYE RENDERS OBVIOUS CLAIMS 1-9, 11-22, 24-35, 37-39

59. Ground 1 proposes the relational database structures of Althoff modified to include (1) the event model taught by Nye —in particular, the database structures taught by Nye for defining and storing record state changes. As explained in detail below, a POSITA would have found this combination obvious, and would have been motivated to combine these teachings at the time of the alleged invention.

1. A POSITA Would Have Been Motivated To Combine Motivation to Combine Althoff and Nye

60. Althoff and Nye each disclose a data model for a database system. Althoff specifically teaches translating data and relationships from the OO model to a relational database. Nye specifically teaches an event model to

add complex dependency logic to an existing database including relational databases, OO databases, or other database environments. Nye's event model comprises 8 database tables that enable a system to model a workflow or process with multiple steps, commonly referred to as a state machine. Nye teaches one embodiment where "customer" objects go through a "6 month free subscription" process when signing up for a newsletter. Ex. 1005, 7:23-36. Nye's teaches the customer objects have a "current status" of "free trial subscription[]," "paying subscriber[]," or "canceled subscription[]." *Id.*

61. In my opinion, a POSITA would have found it obvious to incorporate Nye's teachings of an event model into Althoff for the several reasons discusses below.

62. Like Althoff and Nye, the '468 Patent is directed to the field of computer data storage, more specifically storing and managing data sets in a database system. Ex. 1001, 1:61-65. All three references, therefore, are directed to the same field of endeavor, and are analogous art.

63. In my opinion, a POSITA would have been motivated to combine Althoff and Nye by incorporating Nye's event model database structures into Althoff's object oriented database in order to have the complex relationship logic and handle record state transitions that the event model was known to give. Managing state in an OO system is one of the most important

issues and has necessitated entire software design approaches dedicated to solving this problem, such as the State Pattern, which is a commonly used software design pattern for incorporating state machines in OO systems. A POSITA would have felt this same need to find a solution to incorporate state machines into Althoff. Nye discloses exactly such a solution that a POSITA would have been motivated to combine. As Nye itself explains, its event model "allows a developer to add complex dependency logic to an existing database without having to modify the underlying structure of the database." Ex. 1005, Abstract. In particular, by "using an event model," a database can "flexibly handle state transitions," *i.e.* when "the records [in a database] change over time." *Id.*, 5:38-42. In my opinion, since Nye's event model models state transitions, a POSITA would understand that it is a state machine.

64. Nye discloses an event model "which allows a developer to add complex dependency logic to an existing database without having to modify the underlying structure of the database." Ex. 1005, 5:34-37. Nye discloses that the event model "contains one or more events." *Id.*, 10:58-61. Such event model and events are used to "represent[] a set of actions that are optionally contingent upon a condition." *Id.*, 10:62-63. Nye further discloses that events "each have an event state" which "is defined by the developer and varies according to the event that is created." *Id.*, 12:39-41.

65. Nye also discloses a system for implementing event models, events, and event states that "can be instantiated to represent a number of different event models and the corresponding dependencies that interrelate them." *Id.*, 13:34-37. Figure 4, reproduced below, depicts the system for the event model, events, and event states. *Id.*, Fig. 4. Given Nye's disclosure of implementing event models, a POSITA would have been motivated to apply Nye's event modeling technique to an objected-oriented database, such as the one disclosed in Althoff for several reasons.

66. First, it was long-standing practice to integrate state machines such as the one taught by Nye's event model into OO software develeopment systems. Finite state machines, in which a state machine is defined by a list of its states, its initial state, and the inputs that trigger transition between states, date back to the 1950s. The OO data model, where data is stored in the form of objects, dates back to the 1960s and became common in the field of software development in the 1980s. The integration of the state machine model to define and manage a process where an object has a defined state, and transitions between states, referred to as the *State Pattern*, was common practice in the industry at least by the 1990s. A POSITA would understand that the integration of state machines into the OO model provides many benefits. One such benefit is that state machines are a very clear way to

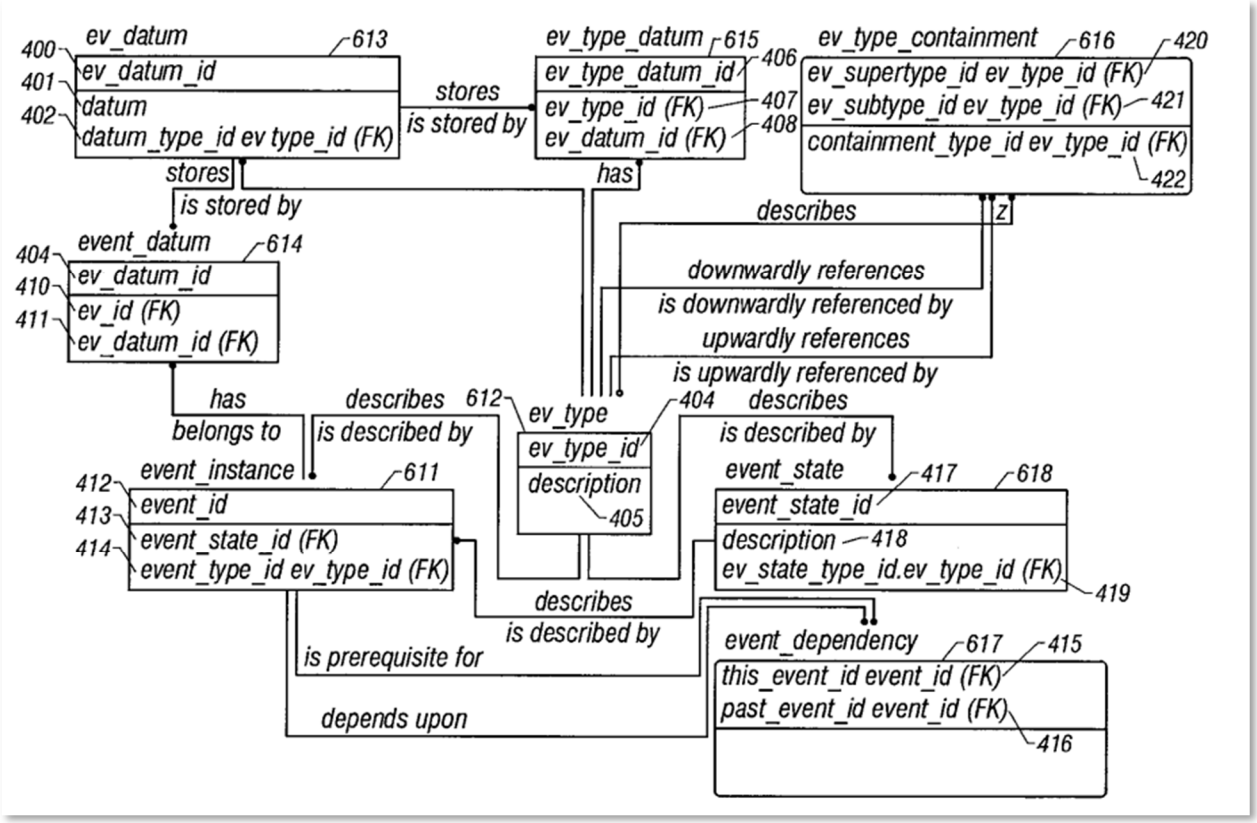
describe behavior of a system. At a high-level, you can describe what the different "modes" of operation of the system are and how it transitions through them. A second benefit is that it is much easier to read a state machine than the underlying software source code. Another benefit of state machines is that they describe the behavior of the system independently of the code. This enables developers to subsequently translate the state machine into code in multiple languages or dynamically change the behavior of the system by changing the model of states. Given the known benefits of state machines, it is unsurprising that they were commonly combined with the OO model.

67. Second, a POSITA would know that integrating Nye's teaching of a state machine into Althoff's relational database would further Althoff's state goal of representing the OO model in a relational model. Althoff taught "it would be advantageous to provide a method of compilation and translation . . .which obtains the advantages of object-oriented database applications, while retaining the ability to use present relational database management systems." Ex. 1008 at 3:4-9. A POSITA would understand that state machines were one such "advantage of [the] object-oriented" model. As noted above, the advantages of state machines include providing a concise, easily readably description of a system's behavior, defining that behavior independent of the code, enabling dynamic changes to the behavior of the

system, and compatibility with multiple programming languages. As discussed herein, in my opinion a POSITA would want to add Nye's event model because it provides the known benefit of a database that can model a process of steps by updating objects stored in the database to reflect a current state of the process, *i.e.* a state machine. Such a benefit was a well-known and desirable feature to add to object-oriented systems, such as Althoff. *Id.*

68. As such, as POSITA would be motivated to combine Nye's event model with Althoff to achieve a set of relational database structures to while maintaining the traditional advantages of the OO model.

69. A POSITA would have known this combination of Althoff and Nye would have reasonably expected to succeed and yield predictable results. The use of Nye's state machine was compatible with Althoff's relational and OO database system, as Nye's state machine was used with "relational databases, object-oriented databases, or other database environments." Nye at 6:41-43. Further, Nye disclosed that the state machine was used "without having to modify the underlying structure of the database." Nye at 6:47-48. A POSITA have understood that implementing Nye's state machine, such as through the relational database structures disclosed in Nye, in Althoff's relational database model would have been expected to succeed. *Id.*



Ex. 1005, Fig. 4.

70. A POSITA would have found it obvious to incorporate Nye's teaching of event models to Althoff's teaching of an OO database to enable modeling of complex relationships and dependencies between data stored in the database. A POSITA would have been motivated to do this in order to achieve benefits of modeling the "corresponding dependencies that interrelate events." *Id.*, 5:34-37. A POSITA would also have been motivated to do this in order to model relationships "without having to modify the underlying structure of the database." *Id.*, 5:34-37.

71. A POSITA would have combined Althoff and Nye with a reasonable expectation of success. The use of Nye's event model was compatible with Althoff's objected-oriented database, as Nye's event model was used with "relational databases, object-oriented databases, or other database environments." *Id.*, 6:41-43. Further, Nye disclosed that the event model was used "without having to modify the underlying structure of the database." *Id.*, 6:47-48. The implementation of Nye's event model was well within the capabilities of a POSITA at the time of the '468 Patent.

72. Moreover, the combination of Althoff and Nye would have yielded predictable results. A POSITA would have readily understood that combining Althoff and Nye would enable event modeling in Althoff's database, and would thus further the functionality of Althoff's system. Further, a POSITA would have understood that this result would be predictable because Nye specifically discloses how to implement the event model in other databases and the results that would be achieved in existing database environments, such as the environment taught by Althoff.

2. Claim 1

(a) 1[pre]: A computerized system for manipulating data sets, comprising:

73. In my opinion, to the extent the preamble is limiting, Althoff discloses a ***computerized system for manipulating datasets***, as claimed.

Althoff teaches the use of "a general purpose ***computer***, or a set of general purpose computers" (*i.e.*, computerized system). Ex. 1006, 5:22-31. Althoff further teaches "allow[ing] a user to create, edit, and ***manipulate a user's object database***." *Id.*, 3:15-18.

74. In my opinion, a POSITA would understand that Althoff's object database includes ***datasets***. An object database is a type of database that stores data grouped into collections called objects. Althoff's object database would thus contain data, which when grouped together is commonly referred to as datasets.

(b) 1[a]: a processor; and

75. In my opinion, Althoff discloses a ***processor***, as claimed. Althoff teaches "using a general purpose ***computer***, or a set of general purpose computers." Ex. 1006, 5:21-31.

76. In my opinion, a POSITA would understand that a general purpose computer includes a central processing unit, also known as a ***processor***. The term "general purpose computer" was a well-known term of art that a POSTIA would understand to include several components, including a processor, memory, input/output devices and a bus connecting these components. Thus, a POSITA would know that Althoff's teaching of a general purpose computer discloses a processor, as claimed.

(c) 1[b]: a data repository, the data repository being adapted to process, retrieve, and store data contained in the data sets and one or more layers of metadata of the data in the data sets, comprising:

77. In my opinion, Althoff discloses a data repository, the data repository being adapted to process, retrieve, and store data contained in the data sets and one or more layers of metadata of the data in the data sets. Althoff discloses the data repository as a "relational database . . . implemented using a relational database engine." Ex. 1006, 6:42-55.

78. In my opinion, A POSITA would understand that a relational database is a specific implementation of a data repository. Relational databases were common in the field of software development at least since the 1980s. A POSITA would have been familiar with relational databases as a software development tool for storing and manipulating data, commonly known in the field as a data repository.

79. Althoff discloses a data repository ***adapted to store data contained in the data sets and one or more layers of metadata of the data in the data sets***, as claimed. Althoff teaches a "user's object ***database***." Ex. 1006, 3:25-29.

80. In my opinion, a POSITA would understand that an object database ***contains data sets*** in the form of objects and the data associated with

those objects. Object databases, also known as object oriented databases, are a type of database that was commonly known in the field of software development by the early to mid 2000s. A POSITA would be familiar with object databases and understand that object databases group data into collections, known as objects, and store the data in a data repository.

81. Althoff further teaches "a ***meta-model of the user's object database***, which is itself an object database [and which] comprises a set of classes, objects, and relationships between classes which model the classes and relationships between classes of the system." Ex. 1006, 3:25-32.

82. A POSITA would understand that a meta-model of the user's object database would contain one or more layers of metadata of the data in the data sets because the meta-model includes data about the data sets (i.e., a layer of metadata). Metadata is data describing other data. For example, a dataset may contain information listing a set of alphanumeric billing codes associated with medical procedures that were performed on a patient. There may be many different alphanumeric billing code standards that use different codes for the same procedure. Thus, metadata can be included in the database that describes for each list of alphanumeric billing codes, which standard the billing codes come from.

83. In this example, the data is the list of billing codes for the medical procedures and the description of which billing code standard the list uses is metadata. Similarly, Althoff's user object database contains data collected and stores as objects. Althoff's meta-model contains data that "comprises a set of classes, objects, and relationships between classes." *Id.* Classes and relationships between classes contain information about the set of objects stored in the underlying user object database. Thus, Althoff's meta-model contains one or more layers of metadata of the data in the data sets, as claimed.

84. Althoff discloses a data repository adapted to process, retrieve, and store data contained in the data sets, as claimed. Althoff teaches that it "allows a user to create, edit and manipulate a user's object database (which the method and system dynamically translates into a set of relational database structures)." Ex. 1006, 3:15-21. Thus, a POSITA would understand that Althoff's relational database is a data repository that ***processes and retrieves data*** stored in the user's objects database (*i.e.*, ***data contained in the data sets***), by enabling the user to create, edit, and manipulate the object database. Further, Althoff's relational database ***stores data contained in the data sets*** by translating the user's object database into a set of relational database structures. *Id.*

85. Althoff discloses a data repository ***adapted to process, retrieve, and store one or more layers of metadata of the data in the data sets***, as claimed. As described above, Althoff teaches a meta-model of the user's object database that includes ***one*** or more layers of metadata of the data in the data sets. Althoff further teaches that the meta-model "may be ***manipulated in the same manner as the user's object database***." Ex. 1006, 3:41-44. Thus, a POSITA would understand that Althoff's relational database (*i.e.*, data repository) is adapted to ***process, retrieve, and store one or more layers of metadata of the data in the datasets*** in the same manner as described above with respect to the data contained in the data sets because the meta-model stores meta-data for objects in the user's object database.

(d) 1[c]: an element module configured to store and uniquely identify a plurality of elements, wherein each of the elements is identified by a unique element identification;

86. Althoff discloses ***an element module configured to store a plurality of elements***, as claimed. Althoff teaches representing classes and objects from an object oriented database within a relational database by explaining that "each class . . . is modeled by a relational database table . . . and ***each object that class is modeled by a row in that table***." Ex. 1006, 3:45-46. As discussed below, Althoff describes a system of classes that are defined in a "class Class" table. For each of these classes, "the system 200 creates a

table in the corresponding relational database 203 to correspond to the selected class 101." *Id.*, 23:10-12. Each table is linked to their particular class in the "class Class" table with a column containing the "class identifier" for each object. *Id.*, 23:33-34.

87. Althoff further teaches that these individual class tables ***store objects*** (*i.e.*, ***elements***) by explaining that "[w]hen an object is populated into the corresponding relational database 203, the system 200 will create a row in the table corresponding to the object's class 101 [and thus] ***each object will correspond to one row in each table*** corresponding to each class 101." *Id.*, 23:19-25.

88. In my opinion, a POSITA would understand that Althoff's individual class tables disclose ***element modules*** because they store objects (*i.e.*, elements) for each class.

89. Althoff discloses ***an element module configured to uniquely identify a plurality of elements, wherein each of the elements is identified by a unique element identification***, as claimed. Althoff teaches that, for each of the individual class tables, "[t]he first column in each table is for the ***UID for each object*** [and thus], in each table in which a row is created for the object, the first column corresponds to the UID for the object." Ex. 1006, 23:26-29; *see also* 3:50-52 ("Each such table comprises an 'object-ID'

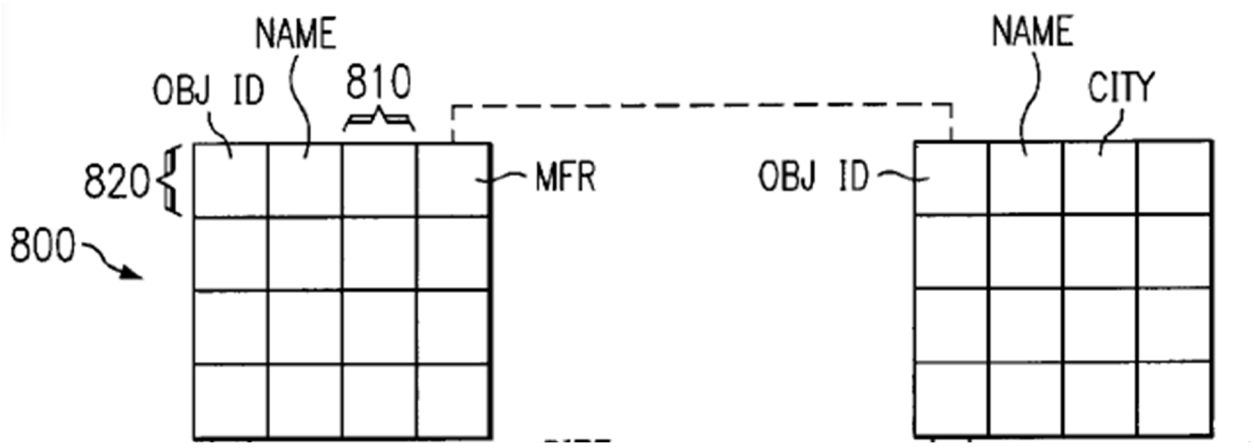
column generated by the system, comprising a unique ID for each object in the system."). Althoff explains that the "UID" is a "***unique identifier*** . . . for each object." *Id.*, 23:14-15. In other words, each object (*i.e.*, ***element***) in the individual class table (*i.e.*, ***element module***) has a unique identifier (*i.e.*, ***is identified by unique element identification***).

(e) 1[d]: an element relation module configured to store one or more relationships between the elements in the element module;

90. Althoff discloses an ***element relation module configured to store one or more relationships between the elements in the element module***, as claimed. As discussed above, the system creates a table for each class which is populated with instance objects of that class corresponding to each row of the table. *Supra,* Section XI2(d). Althoff discloses that Figure 8 is an example of those tables. Ex. 1006, 23:66-24:2 ("As described with regard to FIG. 7, the user relational database 230 corresponding to the user's ***object database 100 comprises a set of relational tables 800***, each corresponding to a class 101.").

91. According to Althoff, in Figure 8, "inheritance ***relationships*** are modeled by providing a column 810 'Object ID' in each table 800." Ex. 1006, Fig. 8, 30:10-12; *see also* 23:51-53 ("[T]he system 200 creates a column in the table for each relationship 103 between the class 101 and a related class

101.”). Althoff teaches objects (*i.e.*, ***elements***) as “row 820 in the table 800,” where each class table includes a UID column “OBJ ID” that uniquely identifies each element, and where each class table 800 also includes a “MFR” column that stores UIDs for elements in other class tables. *Id.*, 24:9-31; *see also* 23:53-55 (“In the column for this relationship 103, the system 200 places the UID of the object in the related class 101 which is related to the object in the class 101 for this table.”). Althoff explains that distinct elements in each table can share the same “object UID” to ***store one or more relationships between*** objects (*i.e.*, ***the elements***) as data-model relationships (*i.e.*, ***an element relation module***): “data-model relationships 103 between source and target classes 101 are modeled by providing a column 810 in the source table 800 pointing to an object in the target table 800, i.e., having the object UID of a row 820 in the target table 800.” *Id.*, 30:37-41.



Id., Fig. 8.

92. Althoff teaches an exemplary embodiment that corresponds to the two relational tables shown above in Figure 8: a class Manufacturer table (right), and a class Component table (left), with each table including a column for OBJ ID (left most column) and populated with rows 820 (*i.e.*, elements). *Id.* The dotted line connecting the MFR of an object (*i.e.*, element) of the class Manufacturer table with the OBJ ID of an element of the class Component table "establish[es] the ***relationship 103 between objects*** of the class 101 component and objects of the class 101 manufacturer." *Id.*, 24:26-31.

93. Althoff further teaches a "class Class Link 302," which is also an ***element relation module***. Althoff teaches that in "meta-model 220, each object of class Class 301 may have zero or more parent objects of class Class 301, and may have zero or more child objects of class Class 301" and that these "***relationships 103*** are represented using a cross-link class Class Link 302." *Id.*, 12:28-31. Althoff teaches that "[e]ach object in the class Class Link 302 links a child class 101 to a parent class 101. A child class 101 may have more than one parent class 101 due to multiple inheritance." *Id.*, 12:34-36.

94. In my opinion, a POSITA would understand that since the table depicted in Figure 8 "correspond[s] to a class 101" and contains rows of

"object[s] of a derived class 101," the "class Class Link 302," which defines relationships between classes, also defines ***relationships between the elements*** (which are instance objects of those classes) of the relational table in Figure 8. *Id.*, 24:1-2, 30:10-15. For example, as discussed above, Figure 8's relational table 800 contains a column that represents the relationship between a Component object and a Manufacturer object. This relationship between objects (*i.e.* ***elements***) reflects a class relationship between the Component class and the Manufacturer class, which is a relationship stored in the "class Class Link 302." *See id.*, 6:12-16 ("[T]he class 101 component has a data-model relationship with the class 101 manufacturer ... each object of the class 101 component has a related object of the class 101 manufacturer."). Thus, the "class Class Link 302" ***stores one or more relationships among the plurality of elements stored in*** tables—including ***the element module*** which is defined in Althoff as a specific implementation of a table (*i.e.*, the class tables).

(f) 1[e]: a class module configured to define at least one class of the elements and store the class;

95. Althoff discloses ***a class module***, as claimed. As discussed above, the ***plurality of elements*** stored in the ***elements module*** are instanced objects of a class. *See* Ex. 1006, 3:45-46 ("each class . . . is modeled by a relational database table ... and each object of that class is modeled by a row

in that table."). In other words, Althoff discloses that these classes are ***classes of the plurality of elements*** (for example, a Manufacturer class of the plurality of Manufacturer objects).

96. Althoff teaches that these ***classes are defined and stored*** in a "class Class 301" table (*i.e.* ***class module***). Althoff teaches that "in the meta-model 220, ***each class of the user's object database 100 is modeled*** by an object of class Class 301." Ex. 1006, 11:13-14. Althoff further teaches that these "***classes***" ***of elements*** can be "***defined*** by a user of the system." *Id.*, 1:31-32. Althoff explains that it creates classes (*i.e.*, "class Class" objects) defined by the user by "provid[ing] a capability for the user 201 to select values for the properties 102 of the new object of ***class Class 301 which has just been created***." *Id.*, 28:28-52.

97. Althoff discloses that these classes and their definitions are ***stored*** in the "class Class 301" table, where "each class ... is modeled by an object [*i.e.* row] of class Class 301" and "each object of class Class 301 comprises information in table 3-2." *Id.*, 11:12-16. Such information includes the "name to be displayed for the class" and a "description of the class." *Id.*, 11:18-48.

(g) 1[f]: an attribute module configured to define one or more attributes and store the attributes;

98. Althoff discloses ***an attribute module configured to define one or more attributes***, as claimed. Althoff teaches that searchable properties define information about a class: "a user's object database 100 comprises a set of classes 101, each comprising a ***set of searchable properties 102, and each representing a type of data*** for an object oriented database to store, manipulate, or retrieve." Ex. 1006, 5:39-42.

99. In my opinion, attributes are a well-known term of art in the OO software that a POSITA would understand. OO software uses objects and classes as key building blocks. *Supra* Section IX. Attributes are collections of data that are associated with both classes and objects. An attribute represents a piece of data associated with the class or object, such as the state the object is in (e.g., "the engine is off").

100. In my opinion, a POSITA would further understand that Althoff's ***searchable properties are the one or more attributes defined*** for each class because they comprise a collection of data associated with each class. *Supra,* ¶99. Indeed, Figure 3 of Althoff teaches that a "Class Property" object "is an attribute of" a "Class" object. *Id.*

101. Althoff additionally discloses an ***attribute module configured to store the attributes***, as claimed. Althoff teaches "Class Property 303" and "Pointer Class Property 304" tables that ***store the attributes***: "[i]n the meta-

model 220, searchable properties comprising built-in data types are represented by a class Class Property 303, while searchable properties comprising user-defined data types (implemented by pointers to objects of the user-defined data types) are represented by a class 101 Pointer Class Property 304." Ex. 1006, 13:2-7. ***Attributes are stored*** within the Class Property and Pointer Class Property classes as objects of the respective classes.

102. For example, "[t]he class Pointer ***Class Property 304 includes one object for each searchable property 102*** for any object of class Class 301 having a user-defined data type, and each object of class Class 301 may have zero or more associated objects of class Pointer Class Property 304. Each object of class Pointer Class Property 304 has a pointer to an object of the user-defined type; the pointer is an object of class Pointer Valid Class 305." *Id.*, 13:16-23. Moreover, at a higher level, Althoff discloses that searchable properties (*i.e.*, ***attributes***) are stored in the object database: "set of ***searchable properties 102*** . . . for an object oriented database ***to store***." *Id.*, 5:39-42.

(h) 1[g]: a class attribute module configured to define and store one or more class-attribute associations between at least one of the attributes and the class;

103. Althoff discloses ***class-attribute associations between at least one of the attributes and the class***, as claimed. Althoff discloses relationships between classes, referred to as class objects, and attributes, referred to as

searchable properties of class objects. Specifically, Althoff teaches that "class Class 301 has a relationship 103 ... which ***links a class 101 to its searchable properties***" (*i.e.*, ***attributes***). Ex. 1006, 13:8-9.

104. In my opinion, a POSITA would understand that relationships (*i.e.*, "relationship 103") in OO programming databases encompass both associations and inheritance relationships. In OO programming a "relationship" is a well know term of art. It describes a link between two entities, referred to as objects, and includes several types of relationships such as association, inheritance, composition, aggregation, dependency, and more. Inheritance describes "is a" relationships between objects, such as a "F150" is a "Ford Truck". In an inheritance relationship one class is either a more generalized or more specific version of the another class. Inheritance relationships are an "is-a" relationship. In contrast, an association occurs between two different objects and does not imply that the objects are of the same type. Association is a more generic type of relationship that describes other classes or objects that a class or object uses, has, or contains.

105. Thus, Althoff's "relationship 103" which "links a class 101 to its searchable properties" (*i.e.*, attributes) discloses ***one or more associations***

(*i.e.*, a type of relationship) ***between at least one of the attributes and the class*** for linking a class to its searchable properties.[14]

106. Althoff's "relationships 103" is ***a class attribute module configured to define and store*** one or more class-attribute associations, as claimed. Althoff teaches ***relational database structures*** comprising "a set of classes 101, ***a set of searchable properties for each class 101***, and a set of relationships 103 between classes 101." *Id.*, 10:12-16. The association between a class (*i.e.*, class Class) and its attributes (*i.e.*, class Class Property and class Class Pointer Property) is defined as a "***data-model relationships 103 between classes 101***." *Id.*, 30:9.

107. In my opinion, a POSITA would understand that the data model relationships between classes and attributes ***define and store the class-attribute associations***. In a relational database the data model describes association among entities stored in the database. For example, a relationship may exist between a supervisor and employees, in that a supervisor can oversee many employees and each employee may be report to one or more

[14] This understanding—that Althoff's disclosure of "relationship 103" encompasses both associations and inheritance relationships—applies throughout this Petition for claim limitations that rely on Althoff.

supervisors. To describe such relationships, data models use three types of relationships: one-to-many, many-to-many, and one-to-one. Data model relationships utilize these three types of relationships to represent relationships such as associations and inheritance.

108. Althoff also discloses a "class Pointer Valid Class 305," which is also a ***class attribute module***. The "class Pointer Valid Class 305" "links a class 101 to its searchable properties with user-defined data types." Ex. 1006, 13:7-9; *see also* 14:34-36 ("Each object of class Pointer Valid Class 305 links an object of class Pointer Class Property 304 to a target class 101."). Specifically, the "class Pointer Valid Class 305" has relationships with both "class Class 301" and "class Pointer Class Property 304." *Id.*, 13:10-14. Thus, the "class Pointer Valid Class 305" ***defines and stores associations between a class and its attributes***.

(i) 1[h]: a type definition module configured to define and store one or more types of the class, the attributes related to the class, and the relationships between the elements;

109. Althoff discloses ***a type definition module configured to define and store one or more types of the class***, as claimed. Althoff's Class Type, Property Type, and Object Type columns are the ***type definition module***. As described above, Althoff's class Class table defines and stores classes of the elements. *Supra*, Section XI2(f). The class Class table contains a Class Type

column which ***defines and stores one or more types of the class*** by indicating "whether the class is part of the meta-model, a cross-link class, a base class . . . or any other class created by the user." Ex. 1006, Tab. 3-2.

110. Althoff discloses ***a type definition module configured to define and store the attributes related to the class***, as claimed. As described above, Althoff's class Class Property defines and stores the attributes related to the class. *Supra*, Section XI2(g). Althoff further teaches that the class Class Property contains a column named Property Data Type (*i.e.*, a type definition module) that ***defines and stores the attributes related to the class*** by "indicat[ing] data type this property is stored as." Ex. 1006, Tab. 3-6.

111. Althoff discloses ***a type definition module configured to define and store the relationships between the elements***, as claimed. As described above, the class tables store elements and relationships between the elements. Althoff teaches that each class table has a set of "specified columns" which include "a column for the 'object type' of each object." *Id.*, 23:13-16. The "the 'object type' column indicates whether this particular object (as opposed to an entire class)" is of a given type. *Id.*, 23:40-42.

112. In my opinion, a POSITA would understand that determining whether or not elements are of the same type ***defines the relationships between the elements, which is stored*** in the "object type" column. In object

oriented programming, objects (i.e. elements) have a type which can have an inheritance relationships with one or more classes and describes relationships between the objects. The object type contains metadata about the object (i.e. element), including the relationship between the object and other objects (i.e. elements).

113. Alternatively, Althoff's class Pointer Valid Class discloses ***a type definition module configured to define and store one or more types of the class, the attributes related to the class; and the relationships between the elements***, as claimed.

114. Althoff's class Pointer Valid Class discloses ***a type definition module configured to define and store one or more types of the class***. The class Pointer Valid Class contains a field "Valid Class Description" that ***defines and stores*** **"**the relation between the pointing and target classes" Ex. 1006, Tab. 3-5. Althoff teaches that the Valid Class Description field enables the user to specify the nature of the relationship between classes. *Id.* Althoff teaches a relationship between classes with the Valid Class Description of "is manufactured by" that ***defines and stores the types of the*** "pointing and target" ***classes*** as type "manufactured part" and "manufacturer" respectively. Ex. 1006, Tab. 3-5. Thus, the class Pointer Valid Class and its Valid Class Description ***define and store one or more types of the class***, as claimed.

115. Althoff's class Pointer Valid Class discloses ***a type definition module configured to define and store the attributes related to the class***, as claimed. As described above, Althoff's Pointer Class Property ***define and store attributes.*** S*upra* Section XI2(g). The Pointer Class Property (i.e. attributes) are ***related to the class*** by Althoff's class Pointer Valid Class which "links an object of class Pointer Class Property 304 to a target class 101." Ex. 1006, 14:34-36.

116. Finally, Althoff's class Pointer Valid Class ***defines and stores the relationships between the elements,*** as claimed. As noted above, Althoff discloses elements as objects (i.e. rows) in the class tables. *Supra* Section XI2(c).

117. In my opinion, a POSITA would understand that elements (i.e. objects) have properties of their class, such as type. A POSITA would also understand that defining and storing one or more types of the class and the relationships between classes also ***defines and stores the relationships between the elements*** that belong to those classes. *Supra* ¶112.

118. Thus, by defining and storing class type and relationships between classes, Althoff's class Pointer Valid Class ***defines and stores the relationships between the elements,*** as claimed.

(j) 1[i] a state machine module configured to store one or more state machine types associated with at least one of the elements; and

119. Nye's event model discloses ***a state machine module configured store one or more state machine types associated with at least one of the elements***, as claimed. Nye teaches that the elements, referred to as records in a database, may be updated and that the "***status of records in a database change over time***." Ex. 1005, 5:28-42. Nye refers to a status of a record as the "state" of a record. *Id.* Nye further teaches that the event model "provides a way to flexibly handle state transitions." *Id.* For example, Nye teaches that "if a set of events relating to credit card authorization system is modeled, then the developer might choose to create the following event states: 'authorization approved,' 'authorization declined,' 'hardware error—cannot retry,' 'line is busy—retry later.'" *Id.*, 12:39-46. Thus, Nye's event model is a ***state machine module*** that represents the various event states and the transitions between those states.

120. Nye also teaches ***storing one or more state machine types*** consisting of different collections of event states by explaining that "the event model is flexible and may 'represent anything it is defined to represent.'" *Id.*, Abstract. Further, Nye discloses that the state machine types are ***associated with at least one of the elements*** because the state machine types represent

the collection of possible event states for the status of records (*i.e.*, ***elements***) in Nye's relational database.

(k) 1[j] a status module configured to store one or more statuses of each state machine.

121. Nye's record state discloses ***a status module configured to store one or more statuses of each state machine***, as claimed. As noted above, Nye's event model discloses ***a state machine module*** that includes a collection of possible event states for a record as shown through the example of a credit card authorization process. The ***statuses of each state machine*** are the current event statuses for a given events from the collection of possible event statuses: "[t]he events 301-308 illustrated in FIG. 3 each have an event state 310. An event state 310 is defined by the developer and varies according to the event that is created." *Id.*, 12:39-41. Nye refers to this as the record state (*i.e.*, ***status module***) that changes over time based on state transitions: "[o]ver time the status of a record in the database changes. A record that was once the newest record is no longer the newest record when another record is entered. The ***status of a record is referred to as the records state***." *Id.*, 7:10-13.

122. Nye teaches that the ***record status is stored in the database*** and can be used to categorize elements (e.g., "users") according to their record status: "In one embodiment of the invention the data contained in a database

is categorized according to its status. Referring now to FIG. 8 for example, a database of users 800 can be categorized according to the users 801-806 current status." *Id.*, 7:23-26. As shown below in Figure 8, Nye teaches a ***state machine*** comprised of ***three event states for a user's record status***: "[f]ree trial subscription category 820 . . . paying subscribers category 830 . . . canceled subscriptions category 840." *Id.*, 7:26-37. Thus, Nye's ***record status stores the status of each state machine*** for a given user by storing one of the three event states that correspond to the user.

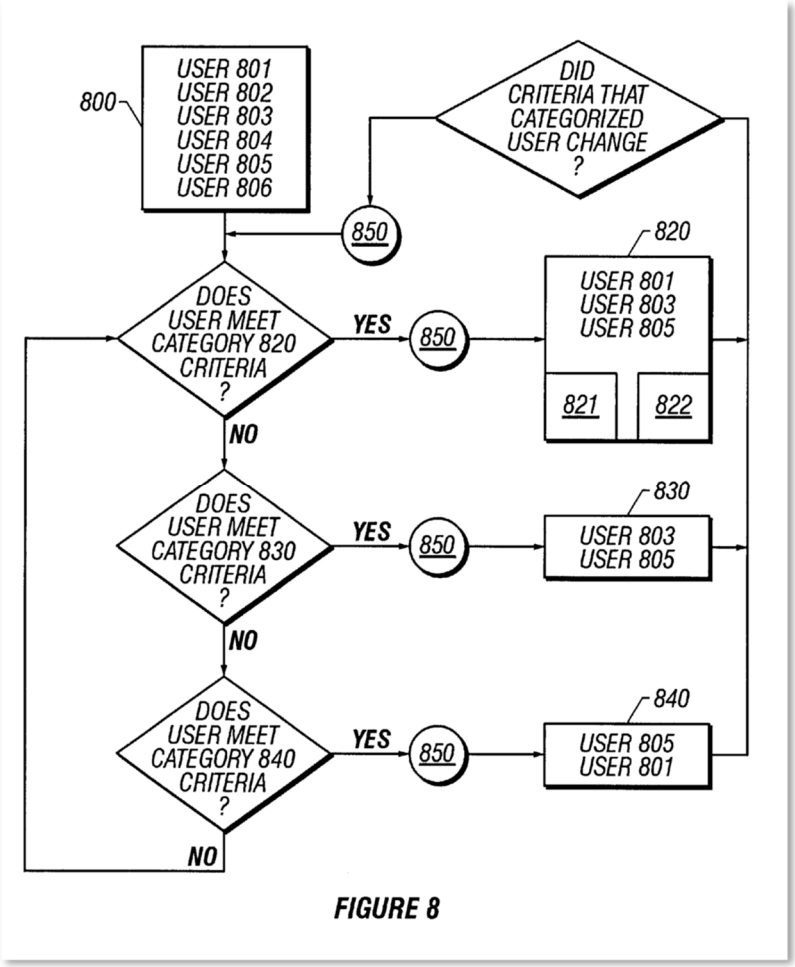


FIGURE 8

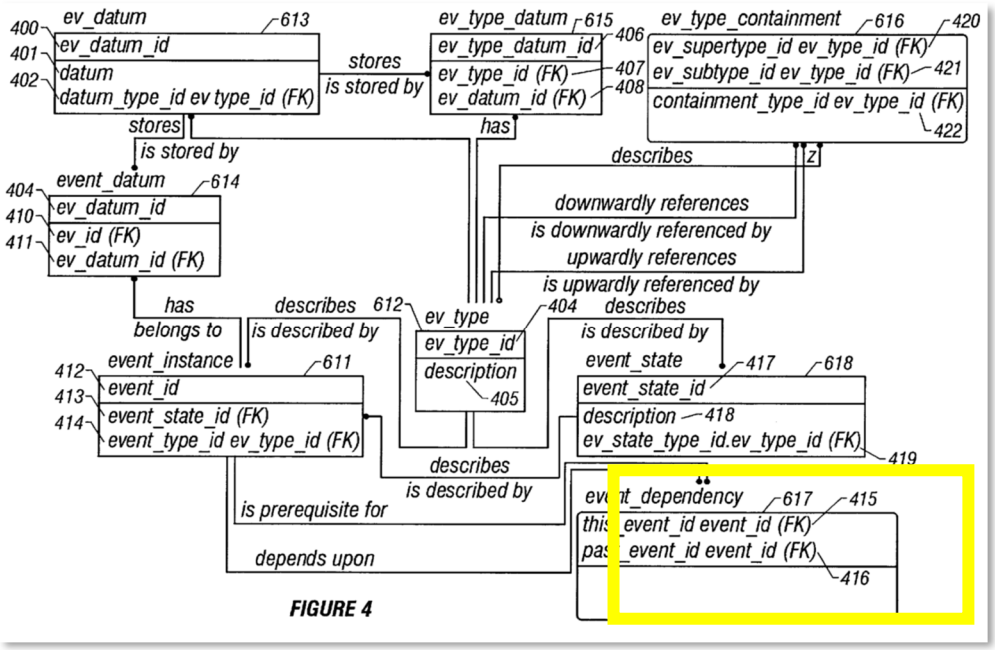
Id., Fig. 8.

2. Claim 2: The computerized system of claim 1, wherein the data repository further comprises: a state machine transition module configured to store a transition from a first status to a second status.

123. Nye's event model discloses ***a state machine transition module***, as claimed. Nye teaches that the event model "***provides a way to flexibly handle state transitions***." *Id.*, 5:28-42. Thus, Nye's state transitions describe the functionality of ***a state machine transition module*** included in the state machine module.

124. Nye's event_dependency table discloses a state machine module ***configured to store a transition from a first status to a second status***, as claimed. Nye's event model teaches a state machine as an event comprised of a collection of event states, and the record status of the event state at a given time as the status of the state machine. Nye teaches that an "event, is a set of actions, possibly empty, which is optionally contingent upon a condition." *Id.*, 9:25-26. Nye teaches that this contingent condition ***defines*** the enter and/or exit conditions for the record status to ***transition from a first status to a second status***: "Event 503, for example, has a condition 532 that causes an action 533 to occur." *Id.*, 10:62-67. The "event_dependency 617 value" ***stores*** the contingent conditions between statuses (*i.e.*, ***stores a transition***) for a given event by "represent[ing] the event dependency relationship of each

event. . . and describes the set of events that another event depends." *Id.*, 14:16-21. As shown below in Fig. 4, the data repository ***stores*** "***event_dependency 617***" ***in the data repository as a table*** in the relational database.



Id., Fig. 4.

3. Claim 3: The computerized system of claim 2, wherein the state machine module defines and stores at least one forward transition connection, the forward transition connection restricting the transition from the first status to the second status.

125. Nye's event trigger functions disclose wherein the state machine module defines at least one forward transition connection, the forward transition connection restricting the transition from the first status to the second status, as claimed. As described above, Nye teaches that the event is

comprised of possible event states for each record at a given time, and the status for each record at a given time. Supra, SectionXI2(j)-(k). As shown below in Figure 5, Nye further teaches that transitions between event states are "optionally contingent upon a condition." Ex. 1005, 10:62-67. For example, "[e]vent 503 . . . has a condition 532 that causes an action 533 to occur." Id.

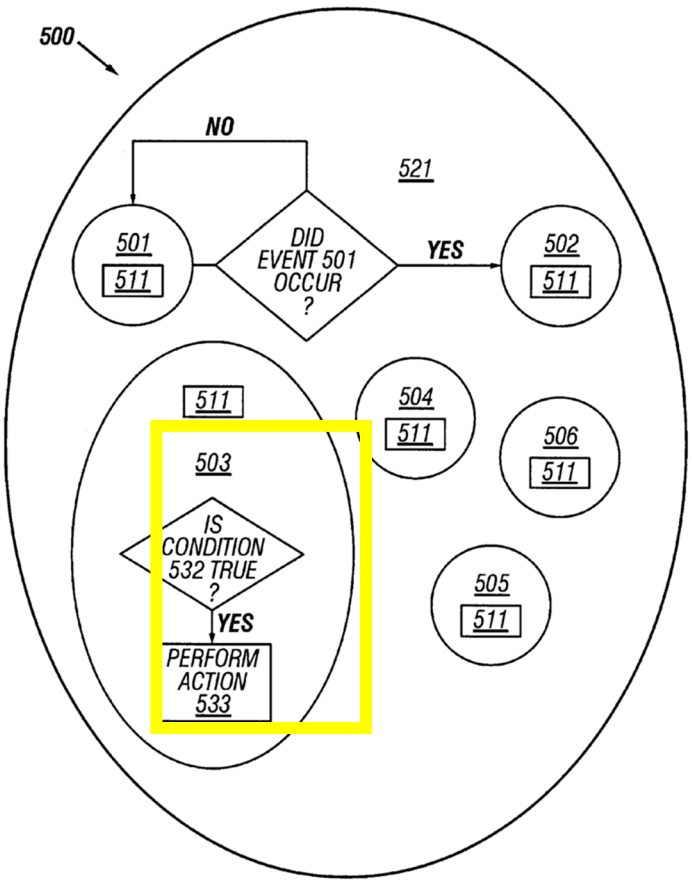


FIGURE 5

Id., Fig. 5. Nye teaches that the action is an "an ***event trigger function*** . . . which takes as input the set of events upon which E depends and their

respective states, and which returns a value of one or zero ***indicating whether the event E is triggered or not triggered***." *Id.*, 15:42-46.

126. Nye further teaches that the event trigger function can restrict the transition of a record status for an event from a first status to a second status: "In one embodiment of the invention the state of an event is computed only if the event is triggered. . . . An event state function is a stored procedure which computes an event_state and event_state_id based on the outcome of the actions it attempts to perform." Id., 15:48-55. Thus, Nye's event trigger function defines a forward transition connection that restricts the transition from the first state to the second state (i.e., status), as claimed.

127. Nye's event functions store at least one forward transition connection, as claimed. Nye's teaches that an event function is a "stored procedure." Id., 15:48-55. In my opinion, a POSITA would understand that the stored procedure stores the event function in the relational database. Stored Procedures are a term of art that would have been well known to a POSITA. A Stored Procedure is a set of Structured Query Language (SQL) statements that access a relational database management system. Stored Procedures are stored in the relational database where they run. Thus, because Nye's event function is a Stored Procedure, it is stored in the relational database.

4. Claim 4: The computerized system of claim 1, wherein the type definition module defines a type including at least one of a boolean type, a picklist type, and a state machine type.

128. Althoff discloses wherein the type definition module defines a type including at least one of a boolean type, a picklist type, and a state machine type, as claimed. As described above, Althoff's class Class Property table discloses a type definition module that defines a type, and Althoff's event model discloses state machine types. Supra, Section XI2(i)-(k).

129. Althoff's class Class Property defines types including at least one of a boolean type, a picklist type, and a state machine type, as claimed. Althoff discloses that the class Class Property table includes a column "Property Data Type" that "indicates data type this property is stored as: boolean, character, date, integer, monetary, numeric, or timestamp." Ex. 1007, 14:47-15:30 (Table 3-6).

130. Further, Althoff's class Class Property table also includes a column "Valid Value Type" that "indicates if value is restricted to certain valid values, e.g., enumerated valid value or range valid value." Id. As noted below, Althoff teaches a picklist as an Enumerated Valid Value object and a Range Valid Value object. Infra, Section XI3-4. By restricting the value of the Property Data Type to a predefined set of choices of an Enumerated Valid

Value or Range Valid Value, Althoff teaches that the class Class Property table defines a type including picklist type.

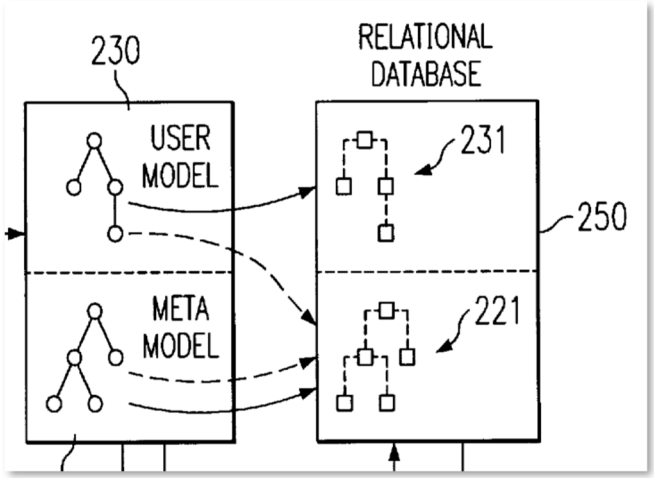
131. A POSITA would understand that this includes ranges for collection of event states defined by state machine types.

5. Claim 5: The computerized system of claim 4, wherein the data repository further comprises: a list option module configured to enumerate one or more picklist attributes of the picklist type.

132. Althoff's Enumerated Valid Value and Range Valid Value each disclose ***a list option module configured to enumerate one or more picklist attributes of the picklist type***, as claimed. Althoff teaches a database record that "may have its value restricted to a specific range or to a specific set of enumerated values" (*i.e.*, ***a list option module***). Ex. 1006, 16:17-18. Althoff's Enumerated Valid Value and Range Valid Value classes include a "a set of enumerated values, or a range of values" (*i.e.*, ***a picklist type***) "as the only valid values for a searchable property 102" (*i.e.*, ***one or more picklist attributes*** of a picklist type). *Id.*, 9:5-11. Althoff further explains that "[w]hen the user 201 so specifies, the system 200 creates an object . . . describing the valid set of enumerated values or the valid range of values, and associates that object with the searchable property 102." *Id.*

133. Althoff discloses that the objects of class Enumerated Valid Value and class Range Valid Value are ***stored in the meta model***: "[i]n the

meta-model 220, a class Enumerated Valid Value 308 includes one object for each set of enumerated valid values . . . . Similarly, a class Range Valid Value 309 includes one object for each set of range valid values." *Id.*, 16:19-27. Further, as shown below in Figure 2, Althoff discloses that the ***meta-model is stored in the relational database***.



Id., Fig. 2.

6. Claim 6: The computerized system of claim 2, wherein the data repository further comprises: a named text module configured to store textual representations of query, control, and display mechanisms that relate to the elements.

134. Althoff's query and access control model discloses a named text module configured to store textual representations of query, control, and display mechanism that relate to the elements, as claimed. Althoff teaches a query model (i.e., named text module) that queries the "user's object database" in response to search descriptions (i.e., textual representations of a query

mechanisms). Ex. 1006, 4:23-27. The search descriptions relates to the objects (i.e., relate to the elements) in Althoff's relational database because they are "to be applied to the relations database structures corresponding to the user's object database." Id.

135. Althoff further teaches that "[w]hen the relational data base query is applied to the relational database, the system receives information supplied from the relational database and presents it to the user according to display properties of the classes which were searched." Id. at 4:38-42.

136. In my opinion, a POSITA would understand that Althoff's display properties, generated in response to a query, are textual representations of display mechanism that relate to the elements which belong to Althoff's classes. There are two types of database queries: an action query and a select query. A select query is one that retrieves data from a database. An action query asks for additional operations on data, such as insertion, updating, deleting or other forms of data manipulation. Both queries pull data stored in the relational database, such as the elements disclosed in Althoff, and display the data as textual representations to the user performing the query.

137. Moreover, Althoff discloses that "***predefined classes***" are used "for ***modeling access control*** to aspects of the user's object database" so that "[a]ccess may be restricted to selected users or selected user groups, and may

be specified for individual objects, classes of objects or properties of those objects." Ex. 1006, 4:42-48. A POSITA would understand Althoff's use of predefined classes for access control is a query ***control mechanism that relates to the elements*** which belong to Althoff's classes. And because predefined classes are used for restricting access to objects, a POSITA would further understand that the access control model belongs to Althoff's query model.

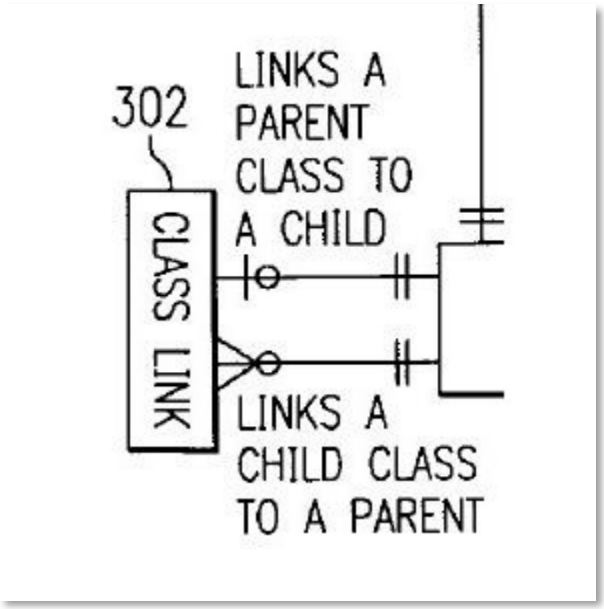
138. Further, in my opinion a POSITA would understand that the combination of queries, access control mechanisms, and display mechanisms are ***stored in*** Althoff's query model (*i.e.*, ***the named text module***) because Althoff discloses "display properties" belonging to "the classes which were searched," "modeling access control," and "generat[ing] a relational database query" in response to a "search description."

7. Claim 7: The computerized system of claim 6, wherein the data repository further comprises: a tuple module configured to generate and store one or more tuples, wherein each of the tuples associates a subset of the elements in accordance to one of the types associated with the subset of the elements.

139. Althoff's Class Link discloses ***a tuple module configured to generate and store one or more tuples***, as claimed. As shown below in Figure 3, Althoff's Class Link class (*i.e.*, ***a tuple module***) is ***configured to store***

objects that link a parent class to a child class (*i.e.*, ***one or more tuples***). Ex. 1006, Fig. 3.

140. A POSITA would understand that Althoff's Class Link class is also ***configured to generate*** these one or more tuples because it provides linking functionality. In relational databases, a tuple is a relationship between two pieces of data or rows in a given database table. Each object of the Althoff's Class Link class is a row in the class link table (i.e. a tuple). Further, relational database tables store data in rows, as claimed.



Ex. 1006, Fig. 3. As shown below in Table 3-3, Althoff further teaches the link between the parent class and child class (*i.e.*, ***one or more tuples***) through the properties Parent Class and Child Class stored for each object (*i.e.*,

element) in the Class Link class. Thus, Althoff's Class Link discloses a tuple module configured to generate and store one or more tuples.

TABLE 3-3

Property	Description
Object ID	unique ID for this object
Parent Class	object ID for parent class
Child Class	object ID for child class
Distributed DB Node	node in a distributed database for which this object is a link

Id., Tab. 3-3.

141. Althoff's objects in the Class Link disclose ***wherein the tuples associate subsets of the elements in accordance with types associated with the subsets of the elements***, as claimed. As explained above, Althoff teaches that elements are objects of a given Class class table. *Supra*, Section XI2. Althoff further teaches that "each class comprises zero or more parent classes (base classes from which that class is derived), and zero or more child classes (derived classes for which that class is a base class). Ex. 1006, 12:25-28. Althoff's objects of Class Link class "***links a child class 101 to a parent class 101***." *Id.*, 12:34-35. Althoff explains that the elements (*i.e.*, objects of class Class) "may have zero or more parent objects of class Class 301, and may have zero or more child objects of class Class 301. These many-to-many

parent/child class relationships 103 are represented using a cross-link class Class Link 302." *Id.*, 12:29-34.

142. In my opinion, a POSITA would understand that a class is a user defined ***type used for associating subsets of*** objects (*i.e.*, ***elements***) that belong to each parent class and child class. Thus, the linking of parent class to child class for the objects of class Class (*i.e.*, the elements) discloses ***tuples that associate subsets of the elements in accordance with types associated with the subsets of the elements***.

8. Claim 8: The computerized system of claim 1, wherein each of the class-attribute associations is identified by a logical name and a name of the class.

143. Althoff discloses ***wherein the class attribute associations are identified by logical names and class names***, as claimed. Althoff's Class Property objects and Pointer Class Property objects are the attributes, and Althoff's class Class objects are the classes. *Supra*, Section XI2. Each Class Property object (*i.e.*, ***attribute***) is ***identified by a logical name*** called Property Label that is the "name to be displayed for the property." Ex. 1006, Tab. 3-6. Similarly, each Pointer Class Property object (*i.e.*, ***attribute***) is ***identified by a logical name*** called Pointer Property Label that is the "name to be displayed for the property," and each Class class object (*i.e.*, ***class***) is ***identified by a***

class name called Class Label that is the "name to be displayed for the class." *Id.*, Tabs. 3-2, 3-4.

144. In my opinion, a POSITA would understand that the names to be displayed for the attributes and classes are the ***logical and class names that identify class attribute associations*** because Althoff discloses attributes associated with objects (*i.e.*, elements) that belong to classes. *Supra*, Section XI2. In software development, a logical name a well know term of art. A logical name is typically a more readable name, such as the name that displays in user interfaces and user-defined applications. Often logical names assist developers and users of the system to identify the underlying data being referred to. Althoff's Property Label fields, which enable the developers to select a name to be displayed for the classes and attributes, are a logical name as claimed.

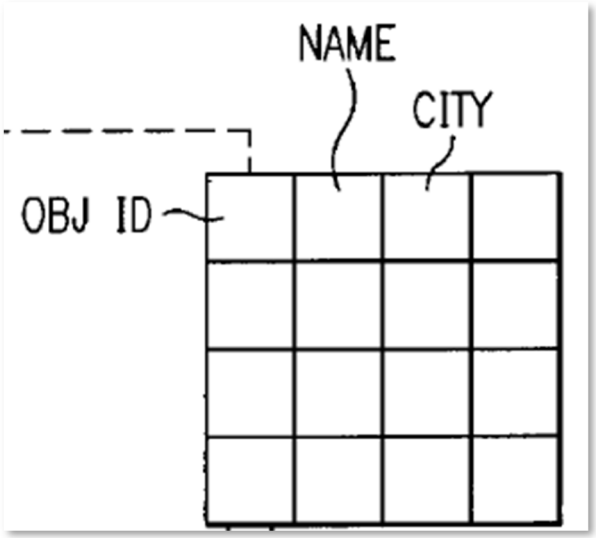
9. Claim 9: The computerized system of claim 8, wherein the data repository further comprises:

(a) an element attribute module configured to define and store the attributes of the elements in the class

145. Althoff discloses an ***element attribute module configured to define and store the attributes of the elements in the class***, as claimed. As explained above, Althoff teaches classes as objects stored in the class Class table, with each class Class object being represented by a separate table

(collectively class tables). *Supra*, Section XI2. Althoff further teaches ***elements as objects stored in the class tables***, where each element is "a row in the table corresponding to the object's class." Ex. 1006, 23:20-21.

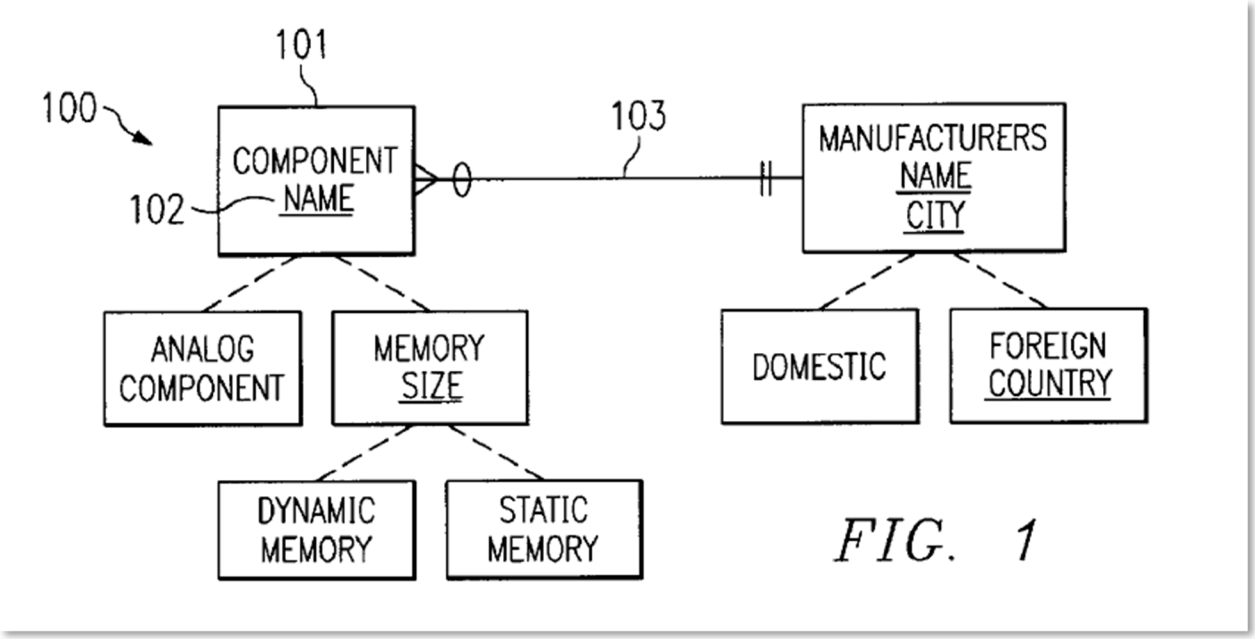
146. Althoff discloses that each class table contains "searchable property" columns that ***define and store*** "a value for a ***searchable property for the class 101***" (*i.e.*, ***attributes***). *Id.*, 23:44-45. For example, Althoff teaches a class named manufacturer, represented by a class table named manufacturer. *Id.*, 6:5-18. As shown below in Figure 8, Althoff's ***class manufacturer includes two attributes***, "***name***" and "***city***," ***for each element of the class manufacturer***, and the manufacturer table includes a column for each of these attributes. *Id.*



Id., Fig. 8.

(b) wherein the element attribute module is identified using the name of the class.

147. Althoff discloses ***wherein the element attribute module is identified using the name of class***, as claimed. As explained above, Althoff's relational class tables are the ***element attribute module***. *Supra*, Section XI9(a). As shown below in Figure 1, Althoff teaches an example with eight classes: component, analog component, memory, dynamic memory, static memory, manufacturers, domestic manufacturer, and foreign manufacturer.



148. Ex. 1006, Fig. 1. Each class is represented by a relational database class table of the same ***name of the classes***: "table 800 component, a table 800 analog component, a table 800 memory, a table 800 dynamic memory, a table 800 static memory, a table 800 manufacturer, a table 800 domestic manufacturer, and a table 800 foreign manufacturer." *Id.*, 23:67-24:8. Indeed, for each "table 800," the system creates "a column 810 'name'."

Id., 24:18-19. Thus, Althoff's relational class tables (*i.e.*, ***the element attribute module***) are each ***identified using the name of the class that they represent***.

10. Claim 11: The computerized system of claim 10, wherein the data repository further comprises: an element document module configured to store one or more formatting data for grouping and displaying a subset of the elements.

149. Althoff discloses an element document module configured to store one or more formatting data for grouping and displaying a subset of the elements, as claimed. Althoff's class Class Configuration is configured to store "one object for each description of a display configuration of a class." Ex. 1006, 18:41-43. Althoff's class tables contain objects (i.e., rows) that are a subset of the plurality of elements. Id., 23:23-25. Althoff's class Class Configuration objects include formatting data for grouping and displaying a subset of the elements, including the "maximum display width" of each element, the "relative order for displaying row[s]" of elements, the "relative order for displaying column[s]" of elements, and a "Custom Label" for displaying the elements. Id., Tab. 3-11.

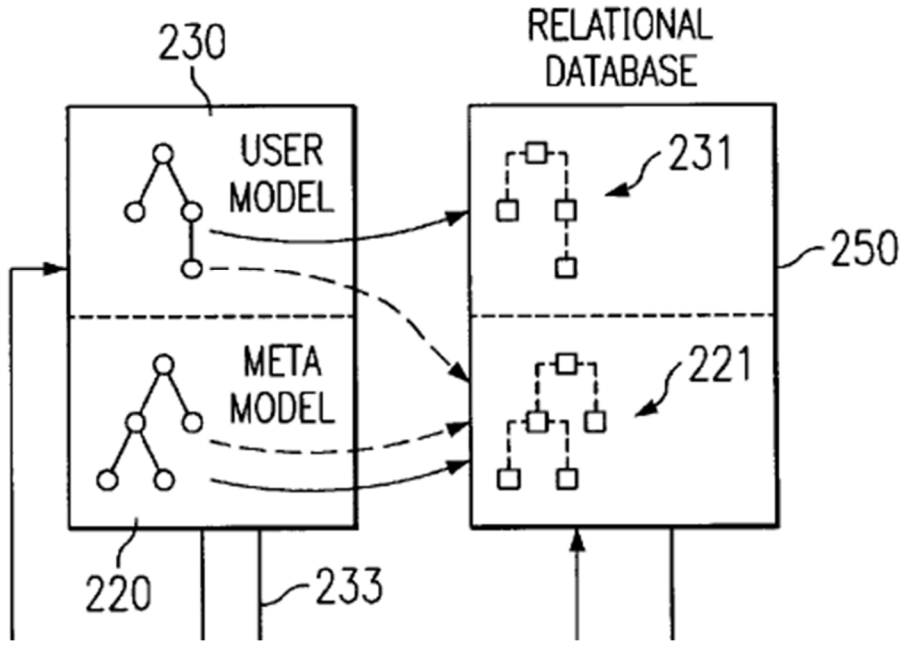
11. Claim 12: The computerized system of claim 1, wherein the processor is configured to generate a data

model of the elements based on the data in the data sets and the layers of metadata stored in the data repository.

150. Althoff's meta-model discloses ***a processor configured to generate a data model of the elements based on the data in the data sets and the layers of metadata stored in the data repository***, as claimed. As described above, Althoff discloses a general purpose computer that includes a hardware-based processor and data sets that are stored in the "user's object database" (i.e., the data repository). Supra, Section XI2(b), Section XI2(c). Althoff's teaches ***the data model*** as "a meta-model of the user's object database, which is itself an object database, and has been translated into a set of relational database structures for manipulation by a relational database engine." Ex. 1006, Abstract. Althoff further discloses ***one or more layers of meta data*** of the objects (i.e., elements) ***stored*** in "a meta-model of the user's object database." Id., 3:25-29.

151. Thus, in my opinion, a POSITA would understand that Althoff's meta-model (i.e., data model) of data sets is generated by the processor based on the data in the data sets and the layers of meta-data, when the object database is translated into a set of relational database structures. Data modeling is a well known term of art that refers to constructing a model of the data stored in a database and layers of meta-data of data in the database. Althoff's user database model 230 "comprises a set of objects in the meta-

model 220, the process of building and editing the user database model 230 simply comprises building and editing objects in the meta-model." Ex. 1006, 6:50-55. Thus the Althoff's metamodel is a data model of the data in the underlying database and the layers of meta-data. The meta-model is ***stored*** as relational database structures: "The user database model 230 is represented by objects in the meta-model relational database 221, comprising rows within those tables, specific values entered in the columns for those rows, and pointers from a row of one table to a row of a related table."



Ex. 1006, Fig. 2

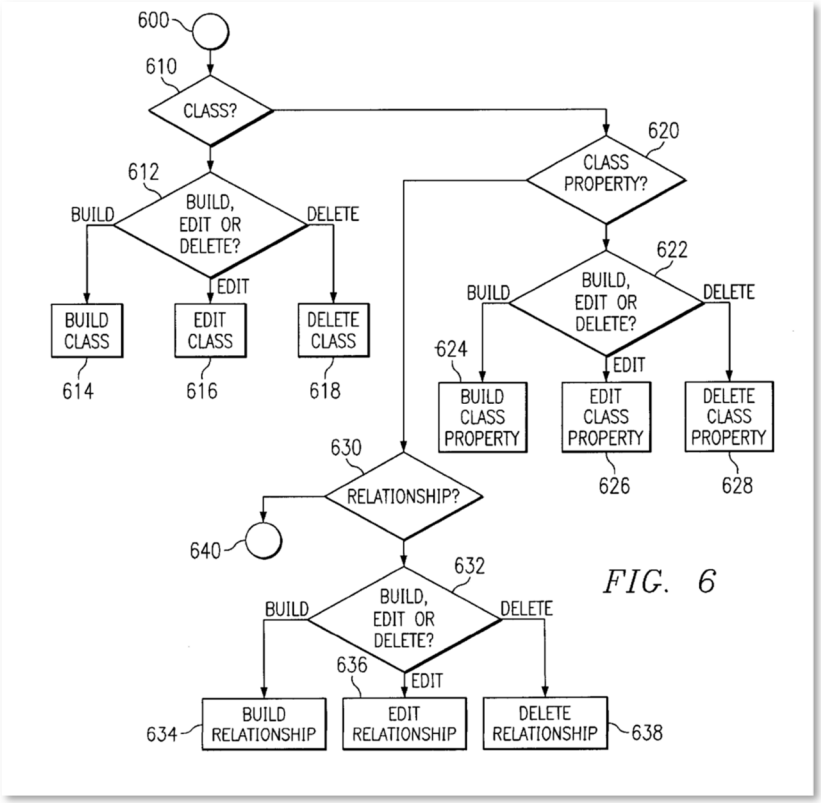
12. Claim 13: The computerized system of claim 1, wherein the processor is configured to generate a dynamic user-interface for accessing and manipulating

the elements based on the data in the data sets and the layers of metadata stored in the data repository.

152. Althoff's discloses generating a dynamic user-interface for accessing and manipulating the elements based on the data in the data sets and the layers of metadata stored in the data repository, as claimed. Althoff's database meta-model stores meta-data (i.e., the layers of metadata) for data sets stored in the relational database (i.e., data repository). Supra, Section XI2(c). Althoff also teaches a Graphic User Interface (GUI) (i.e., a dynamic user interface) that enables accessing and manipulating the elements stored in the database model: "In a preferred embodiment, the user interface 210 comprises a graphical user interface 211 ('GUI'), having editing tools for creating and editing the user database model 230." Ex. 1006, 6:55-63.

153. Althoff further teaches that "[t]he functions provided by the GUI 211 . . . are further described with regard to FIG. 6." *Id.*, 19-22. As shown below in Figure 6, the GUI include functions for ***accessing and manipulating data stored in the database*** through the ability to create, edit, and delete classes, class properties, and relationships.

154. In my opinion, a POSITA would understand that Althoff's GUI is ***based on the data in the data sets and the layers of metadata stored in the data repository*** because the interface displays information about the object-database that is included in Althoff's meta-model.



Id., Fig. 6.

13. Claims 14-22, 24-35, 37-39

155. In my opinion, the limitations of claims 14-22, 24-35, and 37-39 are rendered obvious by Althoff and Nye for the same reasons as discussed in connection with claims 1-9 and 11-13, as set forth in the following chart. *Supra*, Section XI2-12.

Claim Limitation	Corresponding Claim Limitation(s)
14[pre]	1[pre]
14[a]	1[b], 1[c]
14[b]	1[d]

Claim Limitation	Corresponding Claim Limitation(s)
14[c]	1[e]
14[d]	1[f]
14[e]	1[g]
14[f]	1[h]
14[g]	1[i]
14[h]	1[j]
15	2
16	3
17	4
18	5
19	6
20	7
21	8
22	9[pre], 9[a], 9[b]
24	11
25	12
26	13
27[pre]	1[pre]
27[a]	1[b], 1[c]
27[b]	1[d]
27[c]	1[e]

Claim Limitation	Corresponding Claim Limitation(s)
27[d]	1[f]
27[e]	1[g]
27[f]	1[h]
27[g]	1[i]
27[h]	1[j]
28	2
29	3
30	4
31	5
32	6
33	7
34	8
35	9[pre], 9[a], 9[b]
37	11
38	12
39	13

XII. GROUND 2: ALTHOFF IN VIEW OF NYE IN FURTHER VIEW OF SURJANTO RENDERS OBVIOUS CLAIMS 10, 23, 36

1. A POSITA Would Have Been Motivation to Combine Althoff, Nye, and Surjanto

156. Althoff, Nye, and Surjanto each disclose a data model for a database system. Althoff specifically teaches representing data and relationships from the OO model to a relational database. Nye specifically teaches an event model to add complex dependency logic to an existing database including relational databases, OO databases, or other database environments. Surjanto specifically teaches a database management system that improves content management of documents.

157. Like Althoff, Nye, and Surjanto, the '468 Patent is directed to the field of computer data storage, more specifically storing and managing data sets in a database system. Ex. 1001, 1:61-65. All four references, therefore, are directed to the same field of endeavor, and are analogous art.

158. In my opinion, a POSITA would have been motivated to combine Althoff and Nye with Surjanto by incorporating Surjanto's XML document management extensions (*i.e.*, XCoP) with Althoff's OO databases and Nye's event model database structures, because of the well-known benefits of Surjanto's extensions, which were specifically designed to extend OO databases like Althoff's to provide the known benefits of XML document

support, such as "allow[ing] users to reuse and process textual portions of document contents," "enabl[ing] collaborative development of documents" and in general "offer[ing] comprehensive content management functionality." Ex. 1008, 9.

159. A POSITA would expect several benefits from incorporating Surjanto's teachings into Althoff's enables "users to reuse and process textual portions of document contents." Ex. 1008, 9. Further, Surjanto's teachings further "enables collaborative development of documents and facilitates synchronization of fragment modification and versioning." Collectively, Surjanto's teachings integrated with Althoff and Nye would enable users to generate documents from the data stored in Althoff's database structures, and update and format those documents collaboratively. In my opinion, in view of these known benefits, a POSITA would have found it obvious to incorporate Surjanto's teachings into the combination of Althoff and Nye.

160. Surjanto discloses that "[w]ith the advent of object-relational database management systems (ORDBMSs) [Sto98], developers can extend the database system." Ex. 1008, 9. Surjanto "develop[ed] XCoP as part of an integrated Web database framework . . . which is built by our database research group using the extensibility features of an ORDBMS." *Id.* By doing this, Surjanto provides "XML repositories that manage documents at the

content level, allow the flexible definition of content granularity, provide operations on both textual representation and structural information." *Id.* Indeed, Surjanto provides an entire list of benefits from its extensions. *Id.*, 10-11 (describing benefits for "Content Ruse," "Exploitation of Structural Information," "Content Consistency Enforcement," etc.). In other words, Surjanto teaches how to extend OO databases to support storage and manipulation of complex XML documents, both their text and their structure.

161. Indeed, Althoff is exactly the type of object-relational database similar to ORDBMS databases which Surjanto sought to extend. Althoff discloses "object-oriented database structures," including "[a] meta-model of the user's object database," provides "a set of classes, objects, and relationships between classes." Ex. 1006, Abstract.

162. In my opinion, a POSITA would readily understand Althoff's "object-oriented database" is a type of the "object-relational database management systems" disclosed in Surjanto, which describes such databases as "enabl[ing] ... object-oriented data modeling." Ex. 1008, 9.

163. Moreover, Surjanto teaches extending object-relational databases in order to store XML documents. Surjanto teaches database extensions called "XCoP (XML Content Repository," which "manages textual XML document contents together with their structural information."

Ex. 1008, 9. In particular, "XCoP manages both the textual representation of XML document contents ... and the corresponding structural information" by providing "a conceptual model and a set of operations that support the processing of textual contents and structural information." *Id.*, 11. In my opinion, Surjanto teaches how to extend OO databases to support storage and manipulation of complex XML documents, both their text and their structure. And given Surjanto's disclosures relating to ORDBMS databases, a POSITA would have been motivated to apply Surjanto's XML document storage techniques to an OO database system, such as the one disclosed in Althoff.

164. In my opinion, a POSITA would have been motivated to use Althoff's OO database to incorporate these features from Surjanto because at the time of the invention of Althoff, it was commonly understood that "XML is the key technology for the next generation of Web applications and will become the standard format for data representation and data exchange on the Web." Ex. 1008, 10. It was well-known that "[t]he potential impact of XML is obvious" since XML is "a single language for specifying the huge amount of information produced and consumed over the Web" and "provide[s] better interoperation among information providers as well as easier handling of documents by applications." Ex. 1008, 10. In my opinion, one reason a POSITA would have incorporated Surjanto is because it enables developers

to build web applications on top of the Althoff/Nye database. A web application is application software that is accessed over the Internet using a web browser. In the 1990s and early 2000s, the use of web applications was rapidly growing. A POSITA would have been motivated to integrate Surjanto's XML extensions into the combined database system of Althoff/Nye to extend the types of applications that could be built ontop of the database to include web applications. Thus, a POSITA using Althoff's OO database for web applications, would have readily looked to XML, such as disclosed Surjanto, for storage in its database system.

165. In my opinion, a POSITA would have combined Althoff, Nye, and Surjanto with a reasonable expectation of success. Althoff's OO database and database model was compatible with Surjanto, as expressly taught by Surjanto—and the use of Nye's event model database structures would also be compatible. Indeed, Surjanto provides detailed instructions for implementing its extensions into a database like Althoff's as modified by Nye's event model database structures. Ex. 1008, 14-17. The implementation of Surjanto's extensions was well within the capabilities of a POSITA at the time of the '855 Patent, and nothing in Surjanto indicates that a POSITA would have found implementation difficult or unpredictable.

166. Moreover, the combination of Althoff, Nye, and Surjanto would have yielded predictable results. A POSITA would have readily understood that combining Althoff, Nye, and Surjanto would result in a platform for storing the contents of Althoff's database in a web application, and would further allow for interoperability between data from different sources, as expressly taught by Surjanto—while also providing for Nye's representation of event states for the data taught by Surjanto. Indeed, Surjanto explains that this result would be predictable because "XCoP is flexibly configurable." Ex. 1008, 9. Thus, Surjanto would have been readily combined with other database platforms like Althoff and Nye to achieve the predictable benefits taught by Surjanto.

2. Claim 10: The computerized system of claim 9, wherein the data repository further comprises: an element history module configured to store one or more indicators of modification to the elements.

167. Surjanto discloses ***an element history module configured to store one or more indicators of a modification to the elements***, as claimed. Surjanto teaches various ***element types*** referred to as "object types" including "documents" and "fragments." Ex. 1008, 11. Surjanto teaches that "fragments" are "textual portions of document contents." *Id.*, Abstract. Surjanto further teaches that it "enables collaborative development of

documents and facilitates synchronization of fragment ***modification and versioning***." *Id.*

168. In my opinion, a POSITA would understand that a system for fragment versioning corresponds to an element history module configured to store one or more indicators of modifications to different versions of the same fragment (*i.e.*, ***the elements***). A POSITA would understand that when performing concurrent modification of a document, the document can be subdivided into fragments to facilitate different people editing different parts of the document concurrently. Moreover, A POSITA would understand that fragments are versioned so that if users concurrently modify the same fragment or edit different sets of fragments concurrently the history of modifications can be tracked and used to inform all users of changes, rollback changes if there are conflicts, or perform other analyses, such as determining when something was added or removed from the document.

169. Thus, Surjanto's system that facilitates "fragment modification and versioning" discloses an element history module that stores indicators of modification to the elements because a fragment is defined in Surjanto as an object type (*i.e.*, element), and version control requires storing one or more indicators of modifications to each of the fragments.

170. A POSITA would have been motivated to combine Surjanto's disclosure of an element history module with Althoff's relational database to monitor indicators of modifications to the objects (*i.e.*, elements) in Althoff's relational database. *Supra,* Section XII1. This would enable a specific implementation of version control which was a well-known technique at the time of the '468 Patent. *Id*.

3. Claims 18, 20, 23, 31, 33, 36

171. The limitations of claims 23 and 36 are rendered obvious by Althoff, Nye, and Surjanto for the same reasons as discussed in connection with claims 5, 7, and 10, as set forth in the following chart. *Supra*, Section XII2-4.

Claim Limitation	Corresponding Claim Limitation(s)
23	10
36	10

XIII. GROUND 3: ALTHOFF IN VIEW OF NYE IN FURTHER VIEW OF MOORE RENDERS OBVIOUS CLAIMS 6-7, 19-20, 32-33

1. Motivation to Combine Althoff, Nye, and Moore

172. Althoff, Nye, and Moore each disclose a data model for a database system. Althoff specifically teaches representing data and relationships from the OO model to a relational database. Nye specifically

teaches an event model to add complex dependency logic to an existing database including relational databases, OO databases, or other database environments. Moore teaches a system that "is able to take a property that is stored in the database and represent it as a container that is like a folder." Ex. 1007, 3:39-41.

173. Like Althoff, Nye, and Moore, the '468 Patent is directed to the field of computer data storage, more specifically storing and managing data sets in a database system. Ex. 1001, 1:61-65. In my opinion, all four references, therefore, are directed to the same field of endeavor, and are analogous art.

174. In my opinion, a POSITA would have been motivated to combine Althoff and Nye with Moore by incorporating Moore's disclosures of virtual folders of database items with Althoff's relational databases and Nye's event model database structures. A POSITA would be motivated to do this in order to provide the well-known benefit of giving end-users a familiar, folder-based user interface to navigate hierarchical database objects, such as those taught in Althoff (and Nye). This provides a superior experience to the end-user than requiring users to use complicated SQL queries, or navigate complicated tables, in order to view objects stored in Althoff's database.

175. Moore notes that conventional file systems have a number of undesirable features. In general, users of conventional file systems "are generally unable to control the structure that they are shown." Moore at 1:23-25. Another issue is that "a user may have a large number of files which are difficult to organize," especially if files are "in different locations, such as on different devices." *Id.* With conventional file systems, users "have to keep track not only of what file data is stored, but where it is stored." *Id.* at 1:54-58.

176. Thus, Moore discloses a virtual folder system that "is able to take a property that is stored in the database and represent it as a container that is like a folder." *Id.* at 3:39-41. By storing files (and non-file items such as contacts) in a database, and presenting them as virtual folders, "files and folders" are shown to users "based on their metadata instead of the actual physical underlying file system structure on the disk." *Id.* at 3:36-39. A benefit of using virtual folders is that "users are already familiar with working with folders, by presenting the virtual folders in a similar manner, users can adapt to the new system more quickly." *Id.* at 3:41-45. This "virtual folder system 200" comprises a number of components, most notably a "folder processor 210, a relational database 230, [and] a virtual folder descriptions database 232." *Id.*, 8:5-9. One component of the system is a "query builder

component 220 [that] builds SQL queries" from "data from the virtual folder descriptions database 232, including a list of types to display in the folder, the initial filter, and the physical location to show results from (the scopes)."

177. In my opinion, a POSITA would be motivated to use Moore's virtual folder system with Althoff's relational database in order to view items in the database with the familiarity of folders and to separate subsets of the elements into folders, especially where Althoff's relational database is used to store files or other items like contacts, and where Nye's event model database structures discuss collections of event states for objects. In my opinion, a POSITA would have also known that allowing a user to view database items using the familiar folder-based interface would greatly improve a user's ability to visualize and access database objects. A POSITA would understand that a typical database system requires complicated SQL queries to access database information, which would have been beyond the abilities of the average user. Moore teaches a system that automatically generates SQL queries based on user-defined filters and displays the results in a virtual folder system. Thus, for example, an object-oriented database like that in Althoff could store a company's employee directory, and through the teachings of Moore, display it like a folder of employee items, including the use of hierarchical folders for different departments. And even though Althoff

does disclose a GUI for "creating and editing the user database model 230," (Ex. 1006, 6:55-63), Moore's virtual folder system still provides a simpler, and more user-friendly interface that does not require any familiarity with databases.

178. In my opinion, in view of these known benefits, a POSITA would have found it obvious to incorporate Moore's teachings into the combination of Althoff and Nye.

179. In my opinion, a POSITA would have combined Althoff, Nye, and Moore with a reasonable expectation of success. The use of Moore's virtual folder system was compatible with relational databases and database structures, including those disclosed in Althoff and Nye. Moore provides numerous implementation details for incorporating a virtual folder system with databases and database structures, including the use of various components in Moore's Figure 2.

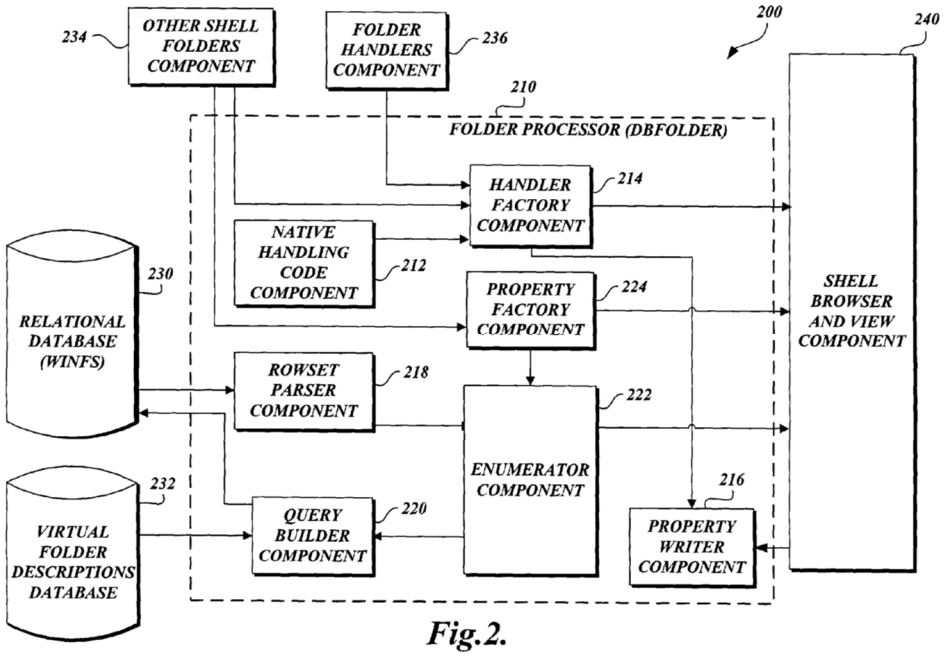


Fig.2.

Ex. 1007 at Fig. 2. The implementation of Moore's virtual folder system into the combination of Althoff and Nye was well within the capabilities of a POSITA at the time of the '468 Patent.

180. A POSITA would understand that Moore was compatible Althoff and Nye's relational database structures. For example, Moore teaches that it's virtual folders include a "folder processor" and that the "folder processor constructs a query. . . passes this query to the relational database. . .the relational database generates the results of the query and passes these back to the folder processor." Ex. 1007 at 10:22-25. A POSITA would understand that Althoff is readily compatible with the virtual folders as it teaches queries "applied to the relational database, the system receives

information supplied from the relational database and presents it to the user." Id. at 4:38-42. A POSITA would understand that the query results could be passed back to the folder processor as taught by Moore to implement its virtual folders.

181. Moreover, the combination of Althoff, Nye, and Moore would have yielded predictable results. In my opinion, a POSITA would have readily understood that combining Althoff and Nye with Moore would result in a database system and database structures as taught by Althoff and Nye with the capability of displaying the objects in the database using a virtual folder system as taught by Moore. Moore teaches that "the system is able to take a property that is stored in the database and represent it as a container that is like a folder." Ex. 1007, 3:39-41. Thus, Moore would have been readily combined with other database platforms, such as Althoff and Nye, to achieve the predictable benefits taught by Moore.

2. Claim 6: The computerized system of claim 2, wherein the data repository further comprises: a named text module configured to store textual representations of query, control, and display mechanisms that relate to the elements.

182. Moore discloses a named text module configured to store textual representations of query, control, and display mechanisms that relate to the elements, as claimed. Moore discloses a "virtual folder descriptions database"

and a "folder processor" (together a named text module) that stores SQL queries (i.e., textual representations of query, control, and display mechanisms) in order to create hierarchical virtual folders for listing/viewing database objects. A virtual folder can "take a property that is stored in the database and represent it as a container that is like a folder." Ex. 1007, 9:66-10:1. Put differently, Moore's system "expose[s] regular files and folders (also known as directories) to users in different views based on their metadata instead of the actual physical underlying file system structure on the disk." Id., 9:63-66.

183. An embodiment of Moore's system is depicted in Figure 2, reproduced below.

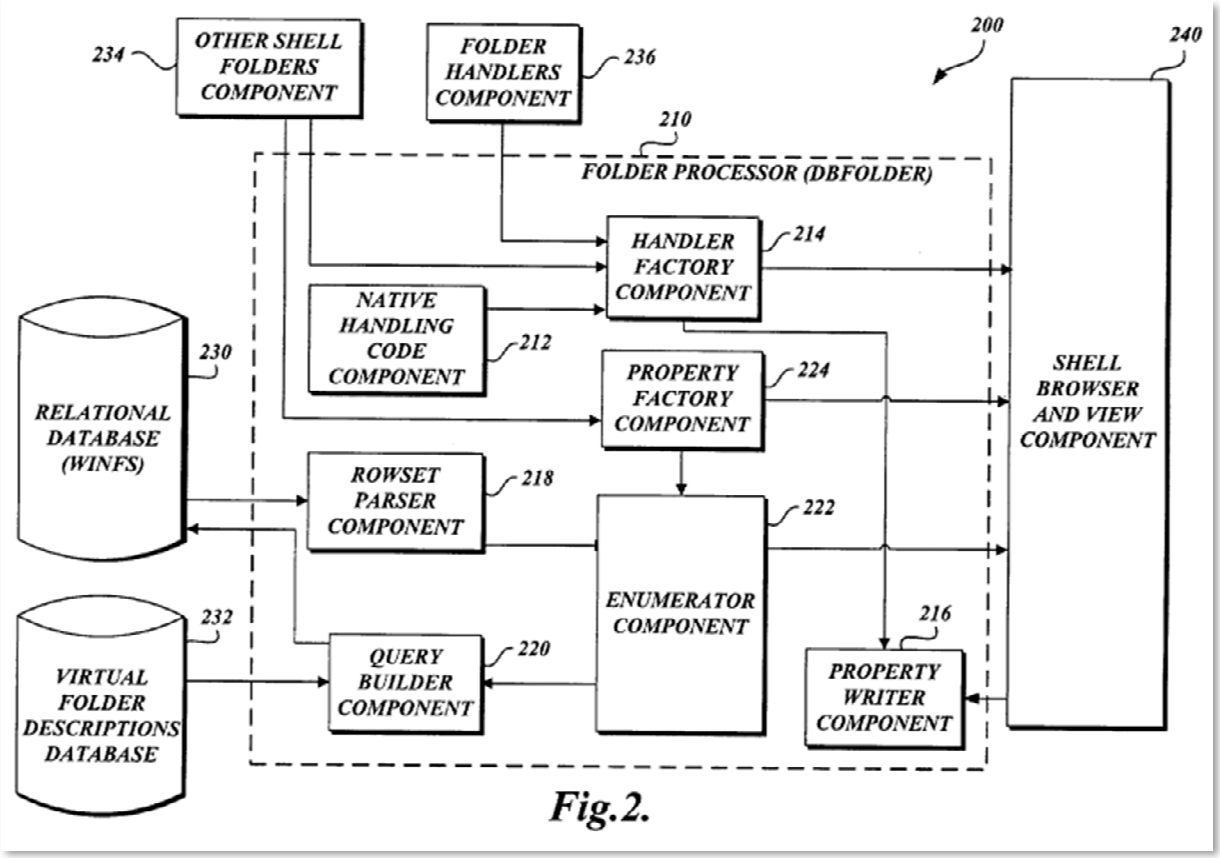


Fig.2.

Id., Fig. 2. As explained by Moore, this "virtual folder system 200" comprises a number of components, most notably a "folder processor 210, a relational database 230, [and] a virtual folder descriptions database 232." *Id.*, 8:5-9.

184. The "virtual folder descriptions database" and "virtual folder system" together make up a ***named text module***. The "virtual folder descriptions database 232 sends data to the query build component 220 [within the virtual folder system], including a list of types to ***display in the folder***, the initial filter, and the physical locations to show results from." *Id.*, 8:24-28.

185. The virtual folder system's "query builder component 220 builds SQL queries" from "data from the virtual folder descriptions database 232, including a list of types to display in the folder, the initial filter, and the physical location to show results from (the scopes)." *Id.*, 9:19-25. The "query builder component 220 sends the SQL queries to the relational database 230." *Id.*, 9:25-27.

186. In my opinion, a POSITA would understand that a SQL query is a "string which can . . . be issued to the database" and results in a "set of rows (in other words a table)." *Id.*, 9:11-13, 9:28-29.

187. A database query is a request for data or information from a database table or combination of tables. There are many database query tools such as Structured Query Language (SQL) which is one of the standard languages used for database management purposes, was invented in the 1970s and SQL queries were well known in the art of relational database systems. SQL queries are strings which request data from an underlying database. SQL queries return data that a POSITA would understand could be a table. SQL queries can also send the data to a another component or system for further processing or displaying the results.

188. Thus, a SQL query is a ***textual representation of query, control, and display mechanisms that relate to the plurality of elements*** because a

SQL query is a string (*i.e.*, ***textual representation***) query for filtering data (*i.e.*, ***plurality of elements***) in a database (*i.e.*, ***control mechanism***) that can also return a table of data for display (or feed the data to another component that displays it) (*i.e.*, ***display mechanism***). Because the query builder component builds SQL queries, it ***stores the textual representations*** (*i.e.*, SQL queries).

189. To the extent Patent Owner argues that the SQL query is not stored because it is dynamically generated, alternatively, the "virtual folder descriptions database 232 includes the virtual folder descriptions" (*i.e.*, ***textual representations***). Ex. 1007, 8:23-24. The virtual folder descriptions constitute "a list of types to display in the folder" and "the initial filter," which are used by the query builder to build the SQL query. *Id.*, 8:24-28, 8:18-20. Thus, since the "virtual folder descriptions" is converted into a SQL query (*i.e.*, ***query, control and display mechanisms that relate to the plurality of elements***), the "virtual folder descriptions" is a ***textual representation*** of those mechanisms.

190. Moore teaches that "[t]he relational database 230 stores properties about all files in the system" but also "non-file items" "like contacts." *Id.*, 8:15-17. The relational database also "stores metadata about the types of files and items that it contains." *Id.*, 8:17-18. When it receives SQL queries, it responds with "SQL rowsets . . . with one row per item column,

columns being the item properties." *Id.*, 8:20-22. Thus, Moore's relational database of file and non-file items is much like Althoff's database of objects, containing ***elements*** (*i.e.*, ***file and non-file items***) and ***attributes*** about those elements (*i.e.*, ***item properties***), and Surjanto's database management system for document fragments (*i.e.*, ***elements***). As already discussed, a POSITA would have known that Moore's virtual folder system of generating SQL queries for displaying of items in a virtual folder could be used with Althoff's database of objects and Surjanto's database management system.

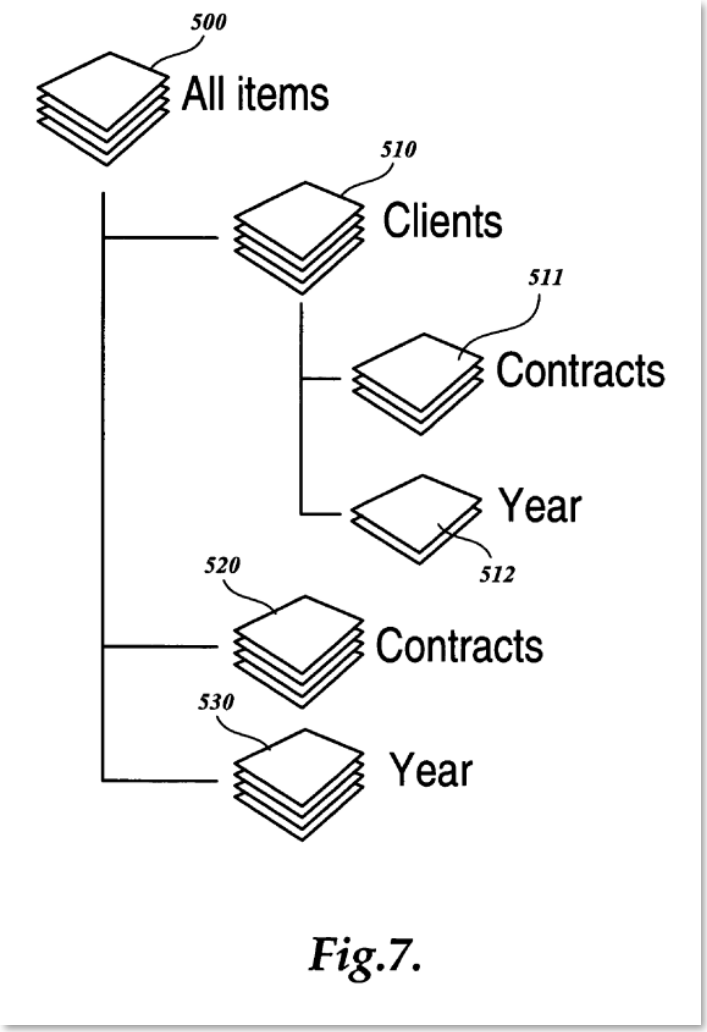
3. Claim 7: The computerized system of claim 6, wherein the data repository further comprises: a tuple module configured to generate and store one or more tuples, wherein each of the tuples associates a subset of the elements in accordance to one of the types associated with the subset of the elements.

191. Moore discloses a tuple module configured to generate and store one or more tuples, wherein each of the tuples associates a subset of the elements in accordance to one of the types associated with the subset of the elements, as claimed.

192. Moore teaches that the "virtual folder descriptions database" (*i.e.*, ***tuple module***) stores "folder descriptions" such as "a ***list of types to display in the folder***, the initial filter, and the physical locations to show results from." Ex. 1007, 22:34-38. Thus, the virtual folder descriptions database links the folder descriptions (*i.e.*, ***folders for each type associated with a subset of the***

elements) with an initial filter (*i.e.*, ***associates a subset of the elements in accordance to one of the types***).

193. Moore teaches that a virtual folders can "further include[] virtual folders." *Id.*, 26:6-7. For example, as depicted in Figure 7 (reproduced below), "[a] virtual folder 500 is an 'all items' folder" that further "contains virtual folders 510, 520, and 530, corresponding to clients, contracts, and year, respectively." *Id.*, 25:66-26:2. Moore further discloses that "at a second level, the virtual folder 510 further includes virtual folders 511 and 512, which correspond to contracts and year, respectively." *Id.*, 26:5-8. Moore depicts and describes several over examples of hierarchical virtual folders that correspond to ***subsets of the elements*** (*i.e.*, clients, contracts, and year) in Figures 6-9.



Id., Fig. 7.

194. Moore also teaches that "the virtual folders explore regular files and folders (also known as directories) to users in different views based on their metadata instead of the actual physical underlying file system structure on the disk." *Id.*, 24:6-9. Since the virtual folder can itself contain folders (and folders can further contain more folders), that results in a hierarchy of

folders for ***subsets of the elements associated with each type*** (since the folders are items in the relational database).

195. Another example of a hierarchy of data entries provided by Moore's virtual folder system is where "a user could view their music" in multiple ways, such as "a flat list of all the songs" or a hierarchical view that "show[s] only the genres or artists or years, etc." *Id.*, 22:2-6.

4. Claims 19-20, 32-33

196. In my opinion, the limitations of claims 19-20 and 32-33 are rendered obvious by Althoff, Nye, and Moore for the same reasons as discussed in connection with claims 6 and 6, as set forth in the following chart. *Supra*, Section XIII2-4.

Claim Limitation	Corresponding Claim Limitation(s)
19	6
20	7
32	6
33	7

XIV. SECONDARY CONSIDERATIONS

197. At this stage of these proceedings, it is my understanding that Petitioner has no burden to identify and rebut secondary considerations. Rather, it is my understanding that Patent Owner must first present a prima facie case for such consideration, which Petitioners should then have the chance to rebut. That said, I have considered evidence of secondary considerations that I am aware of at this time, and I am currently unaware of any evidence of secondary considerations that would have any impact on my conclusions of obviousness as expressed above.

XV. CONCLUSION

198. For at least the foregoing reasons, the instant petition should be instituted.

199. 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 within the United States during the time allotted for cross-examination.

200. I hereby declare that all statements made herein of my own knowledge are true and that all statements made on the 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: June 20, 2023

/s/ Jules White
Jules White, Ph.D.