IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

_____

BEFORE THE PATENT TRIAL AND APPEAL BOARD

_____

SERVICENOW, INCORPORATED,

Petitioner,

v.

INQUISIENT INCORPORATED

Patent Owner.

_____

Case No. IPR2023-01073

U.S. Patent No. 8,224,855

_____

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

TABLE OF CONTENTS

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 ’855 PATENT 9
VII. PROSECUTION OF THE ’855 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 SURJANTO RENDERS OBVIOUS CLAIMS 1, 3-12, 14-23, 25-33..... 26
2. A POSITA Would Have Been Motivation to Combine Althoff and Surjanto..... 26
3. Claim 1..... 31
(a) 1[pre]: A computerized system for manipulating data sets, comprising: 31
(b) 1[a]: a hardware-based processor; and 32
(c) 1[b]: a data repository that contains the data sets and one or more layers of metadata of the data in the data sets, comprising:..... 33

(d) 1[c]: a class module configured to define at least one class of the elements and store the class; 35

(e) 1[d]: an attribute module configured to define one or more attributes and store the attributes; 36

(f) 1[e]: 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;..... 38

(g) 1[f]: an element attribute module for the at least one class configured to: define and store the attributes of the elements in the class; and transmit the attributes and elements;..... 41

(h) 1[g]: an element module configured to receive the transmitted attributes and elements from the element attribute module, store the attributes and elements, and uniquely identify the elements, wherein the elements are identified by unique element identifications; 44

(i) 1[h]: an element relation module configured to store one or more relationships between the elements in the element module; 47

(j) 1[i]: an element history module configured to store one or more indicators of a modification to the elements; and 50

(k) 1[j]: an element document module configured to store formatting data for grouping a subset of the elements. 52

4. Claim 3: The computerized system of claim 1, wherein the data repository further comprises: 53

(a) 3[a]: 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; and 53

(b) 3[b]: a list option module configured to enumerate one or more picklist attributes of a picklist type..... 56

5. Claim 4: The computerized system of claim 1, wherein the data repository further comprises at least one of a named text module configured to store textual representations of query, control, and display mechanisms that relate to the elements..... 58

6. Claim 5: The computerized system of claim 1, wherein the data repository further comprises: a tuple module configured to generate and store one or more tuples, wherein the tuples associate subsets of the elements in accordance with types associated with the subsets of the elements..... 60

7. Claim 6: The computerized system of claim 5, wherein the tuple module uses recursion to link a first folder to at least one additional folder in a hierarchy. 63

8. Claim 7: The computerized system of claim 6, wherein the tuple module includes one or more entries, and the data repository filters the one or more entries into one or more folders..... 65

9. Claim 8: The computerized system of claim 1, wherein the class attribute associations are identified by logical names and class names..... 66

10. Claim 9: The computerized system of claim 1, wherein the element attribute module is identified using a name of the class. 67

11. Claim 10: The computerized system of claim 1, wherein the processor is configured to generate a data model of the elements based on the layers of metadata. 69

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

dynamic user interface for manipulating the elements based on the layers of metadata stored in the data repository. 71

XII. GROUND 2: ALTHOFF IN VIEW OF SURJANTO IN FURTHER VIEW OF NYE RENDERS OBVIOUS CLAIMS 2, 13, 24 75

1. A POSITA Would Have Been Motivated To Combine Motivation to Combine Althoff and Surjanto with Nye 75

2. Claim 2: The computerized system of claim 1, wherein the data repository further comprises: 83

(a) 2[a]: a state machine module including a state machine transition module and a status module configured to define and store a state machine type, ... 83

(b) 2[b]: the state machine type storing a first status of one element and a transition to a second status of the one element. 87

3. Claims 13 and 24 88

XIII. GROUND 3: ALTHOFF IN VIEW OF SURJANTO IN FURTHER VIEW OF MOORE RENDERS OBVIOUS CLAIMS 4, 6-7, 15, 17-18, 26, 28-29 89

1. Motivation to Combine Althoff, Surjanto, and Moore 89

2. Claim 4: The computerized system of claim 1, wherein the data repository further comprises at least one of a named text module configured to store textual representations of query, control, and display mechanisms that relate to the elements. 95

3. Claim 6: The computerized system of claim 5, wherein the tuple module uses recursion to link a first folder to at least one additional folder in a hierarchy. 100

4. Claim 7: The computerized system of claim 6, wherein the tuple module includes one or more

entries, and the data repository filters the one or more entries into one or more folders. 103

5. Claims 15, 17-18, 26, 28-29 104

XIV. SECONDARY CONSIDERATIONS 105

XV. CONCLUSION 105

LIST OF EXHIBITS

Ex.	Description
1001	U.S. Patent No. 8,224,855 ("'855 Patent")
1002	File History for U.S. Patent No. 8,224,855 ("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 hardware-based processor; and
1[b]	a data repository that contains the data sets and one or more layers of metadata of the data in the data sets, comprising:
1[c]	a class module configured to define at least one class of elements and store the class;
1[d]	an attribute module configured to define one or more attributes and store the attributes;
1[e]	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[f]	an element attribute module for the at least one class configured to: define and store the attributes of the elements in the class; and transmit the attributes and the elements;
1[g]	an element module configured to receive the transmitted attributes and elements from the element attribute module, store the attributes and elements, and uniquely identify the elements, wherein the elements are identified by unique element identifications;
1[h]	an element relation module configured to store one or more relationships between the elements in the element module;
1[i]	an element history module configured to store one or more indicators of a modification to the elements; and
1[j]	an element document module configured to store formatting data for grouping a subset of the elements.
2[pre]	The computerized system of claim 1, wherein the data repository further comprises:

Claim	Limitation
2[a]	a state machine module including a state machine transition module and a status module configured to define and store a state machine type,
2[b]	the state machine type storing a first status of one element and a transition to a second status of the one element.
3[pre]	The computerized system of claim 1, wherein the data repository further comprises:
3[a]	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; and
3[b]	a list option module configured to enumerate one or more picklist attributes of a picklist type.
4	The computerized system of claim 1, wherein the data repository further comprises at least one of a named text module configured to store textual representations of query, control, and display mechanisms that relate to the elements.
5	The computerized system of claim 1, wherein the data repository further comprises: a tuple module configured to generate and store one or more tuples, wherein the tuples associate subsets of the elements in accordance with types associated with the subsets of the elements.
6	The computerized system of claim 5, wherein the tuple module uses recursion to link a first folder to at least one additional folder in a hierarchy.
7	The computerized system of claim 6, wherein the tuple module includes one or more entries, and the data repository filters the one or more entries into one or more folders.
8	The computerized system of claim 1, wherein the class attribute associations are identified by logical names and class names.

Claim	Limitation
9	The computerized system of claim 1, wherein the element attribute module is identified using a name of the class.
10	The computerized system of claim 1, wherein the processor is configured to generate a data model of the elements based on the layers of metadata.
11	The computerized system of claim 1, wherein the processor is configured to generate a dynamic user interface for manipulating the elements based on the layers of metadata stored in the data repository.
12[pre]	A method for manipulating data sets, comprising:
12[a]	defining at least one class of elements and storing the class in a data repository that contains the data sets and one or more layers of metadata of the data in the data sets;
12[b]	defining one or more attributes and storing the attributes in the data repository;
12[c]	defining one or more class-attribute associations between at least one of the attributes and the class, and storing the one or more class-attribute associations in the data repository;
12[d]	defining, for at least one class, a set of the attributes of the elements in the class and storing the set of the attributes in the data repository;
12[e]	transmitting the set of the attributes and the elements;
12[f]	receiving the transmitted attributes and elements and storing the attributes and the elements in the data repository;
12[g]	uniquely identifying the elements, wherein the elements are identified by unique element identification;
12[h]	storing one or more relationships between the elements in the data repository;

Claim	Limitation
12[i]	storing, in the data repository, one or more indicators of a modification to the elements; and
12[j]	storing, in the data repository, formatting data for grouping a subset of the elements.
13[pre]	The method of claim 12, further comprising:
13[a]	defining a state machine type that stores a first status of one element and a transition to a second status of the one element; and
13[b]	storing the state machine type in the data repository.
14[pre]	The method of claim 12, further comprising:
14[a]	defining: one or more types of the class; the attributes related to the class; and the relationships between the elements; storing the types in the data repository; and
14[b]	enumerating one or more picklist attributes of a picklist type and storing the enumerated picklist attributes in the data repository.
15	The method of claim 12, further comprising: storing, in the data repository, textual representations of at least one of query, control, and display mechanisms that relate to the elements.
16	The method of claim 12, further comprising: generating one or more tuples, wherein the tuples associate subsets of the elements in accordance with types associated with the subsets of the elements; and storing the tuples in the data repository.
17	The method of claim 16, further comprising: using a recursion to link a first folder of the one or more tuples to at least one additional folder of the one or more tuples in a hierarchy.
18	The method of claim 17, wherein the one or more tuples include one or more entries and further comprising: filtering the one or more entries into one or more folders.

Claim	Limitation
19	The method of claim 12, wherein the class-attribute associations are identified by logical names and class names.
20	The method of claim 12, wherein the attributes are identified using class names.
21	The method of claim 12, 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.
22	The method of claim 12, 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.
23[pre]	A non-transitory computer-readable medium storing a computer-executable program which, when executed by a processor performs a method comprising:
23[a]	defining at least one class of elements and storing the at least one class in a data repository that contains data sets and one or more layers of metadata of data in the data sets;
23[b]	defining one or more attributes and storing the attributes in the data repository;
23[c]	defining one or more class-attribute associations between at least one of the attributes and the class;
23[d]	storing the one or more class-attribute associations in the data repository;
23[e]	defining, for the at least one class, attributes of the elements in the class and storing the attributes in the data repository;
23[f]	transmitting the attributes and the elements;
23[g]	receiving the transmitted attributes and elements and storing the attributes and the elements in the data repository;

Claim	Limitation
23[h]	uniquely identifying the elements, wherein the elements are identified by unique element identifications;
23[i]	storing one or more relationships between the elements in the data repository;
23[j]	storing, in the data repository, one or more indicators of a modification to the elements; and
23[k]	storing, in the data repository, formatting data for grouping a subset of the elements.
24[pre]	The computer-readable medium of claim 23, further comprising:
24[a]	defining a state machine type that stores a first status of one element and a transition to a second status of the one element; and
24[b]	storing the state machine type in the data repository.
25[pre]	The computer-readable medium of claim 23, further comprising:
25[a]	defining: one or more types of the class; the attributes related to the class; and the relationships between the elements; storing the types in the data repository; and
25[b]	enumerating one or more picklist attributes of a picklist type and storing the enumerated picklist attributes in the data repository.
26	The computer-readable medium of claim 23, further comprising: storing, in the data repository, textual representations of at least one of query, control, and display mechanisms that relate to the elements.
27	The computer-readable medium of claim 23, further comprising: generating one or more tuples, wherein the tuples associate subsets of the elements in accordance with types associated with the subsets of the elements; and storing the tuples in the data repository.

Claim	Limitation
28	The computer-readable medium of claim 27, further comprising: using recursion to link a first folder of the one or more tuples to at least one additional folder of the one or more tuples in a hierarchy.
29	The computer-readable medium of claim 28, wherein the one or more tuples include one or more entries and further comprising: filtering the one or more entries into one or more folders.
30	The computer-readable medium of claim 23, wherein the class-attribute associations are identified by logical names and class names.
31	The computer-readable medium of claim 23, wherein the attributes are identified using the class names.
32	The computer-readable medium of claim 23, further comprising: generating a data model of the elements based on the layers of metadata.
33	The computer-readable medium of claim 23, further comprising: generating a dynamic user-interface for manipulating the elements based on the data in the data sets and the layers of metadata.

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

Ground	Reference(s)	Basis	Claims
1	Althoff and Surjanto	§ 103	1, 3-12, 14-23, 25-33
2	Althoff, Surjanto, Nye	§ 103	2, 13, 24
3	Althoff, Surjanto, Moore	§ 103	4, 6-7, 15, 17-18, 26, 28-29

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. 8,224,855 (the "'855 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 '855 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 '855 patent. I was a person of at least ordinary skill in the art at the time of the priority date of the '855 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 '855 patent.

VI. OVERVIEW OF THE '855 PATENT

25. U.S. Pat. No. 8,224,855 (the "'855 patent") was filed on June 3, 2011 and issued on July 12, 2011. The '855 patent claims priority to a provisional application filed on June 14, 2005.

26. The '855 patent claims that conventional "data storage systems" provide their own data model, which "lack[s] flexibility and portability." Ex. 1001, 1:22-29. 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 '855 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.

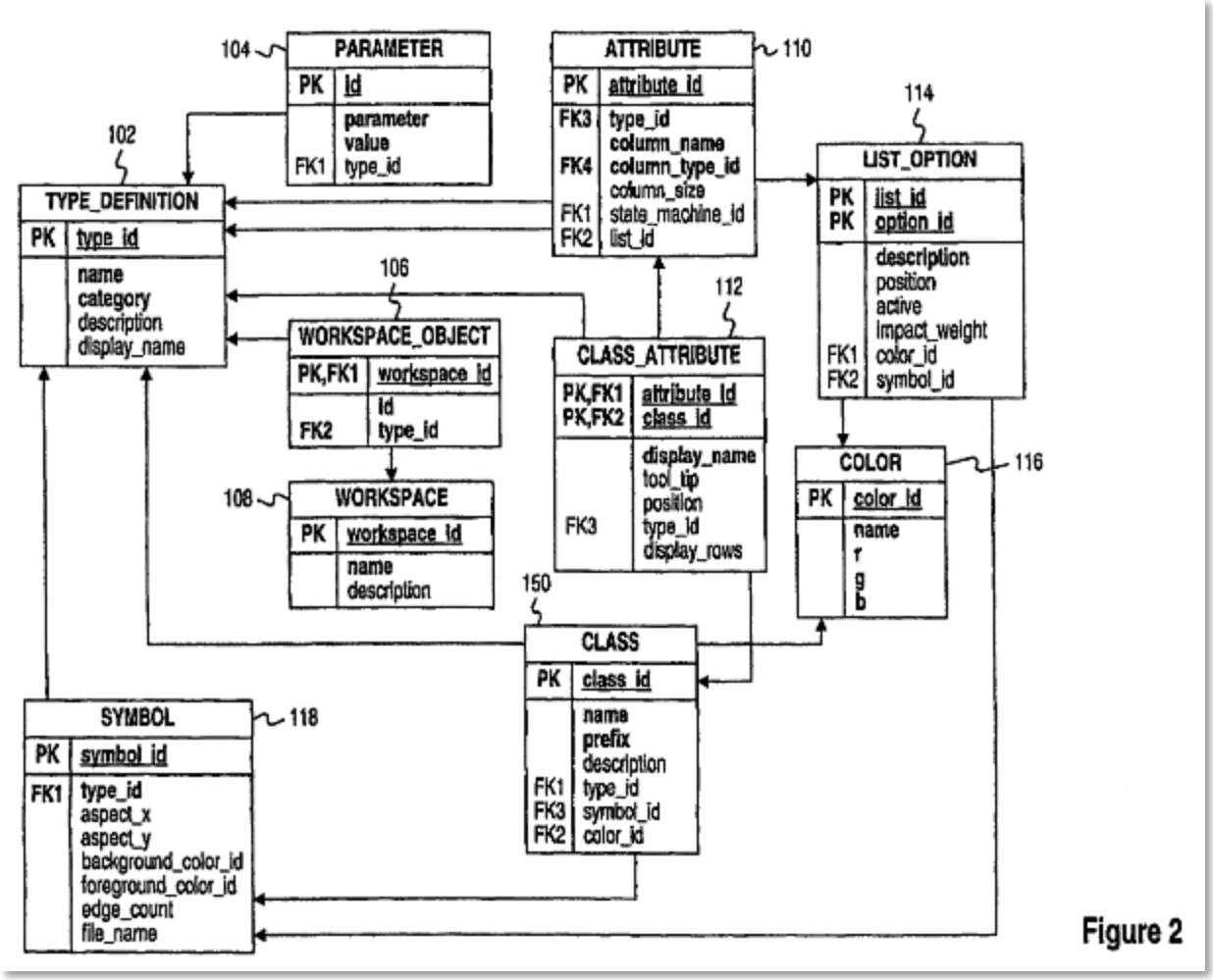


Figure 2

Id., Fig. 2.

VII. PROSECUTION OF THE ’855 PATENT

28. The ’855 patent was filed on June 3, 2011 as U.S. Patent Application No. US13/067,481. The claims as filed issues with little change. The independent claims were amended to overcome prior art in a March 16, 2012 examiner amendment to add limitations to independent claims 1, 12, and 23, specifically the “element history module configured to store one or more indicators of a modification to the elements” and the “element document

module configured to store formatting data for grouping a subset of the elements." Ex. 1002, Examiner Amendments (March 16, 2012).

VIII. CLAIM CONSTRUCTION

29. 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. With the exception of "module," 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.

30. 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. The '855 patent also describes the "data repository including" each of these "modules." A POSITA would thus understand "module" to be a portion of a data repository. I apply Patent Owner's own definition of "module."

IX. BACKGROUND TECHNOLOGY

A. Object Oriented Software Development Models

31. 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.

32. 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.

33. 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".

34. 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.

35. 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.

36. 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.

37. 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.

38. 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.

39. 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.

40. 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.

41. 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.

42. 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.

43. 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)

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

45. 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)

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

47. 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

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

49. 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.

50. 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.

51. 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.

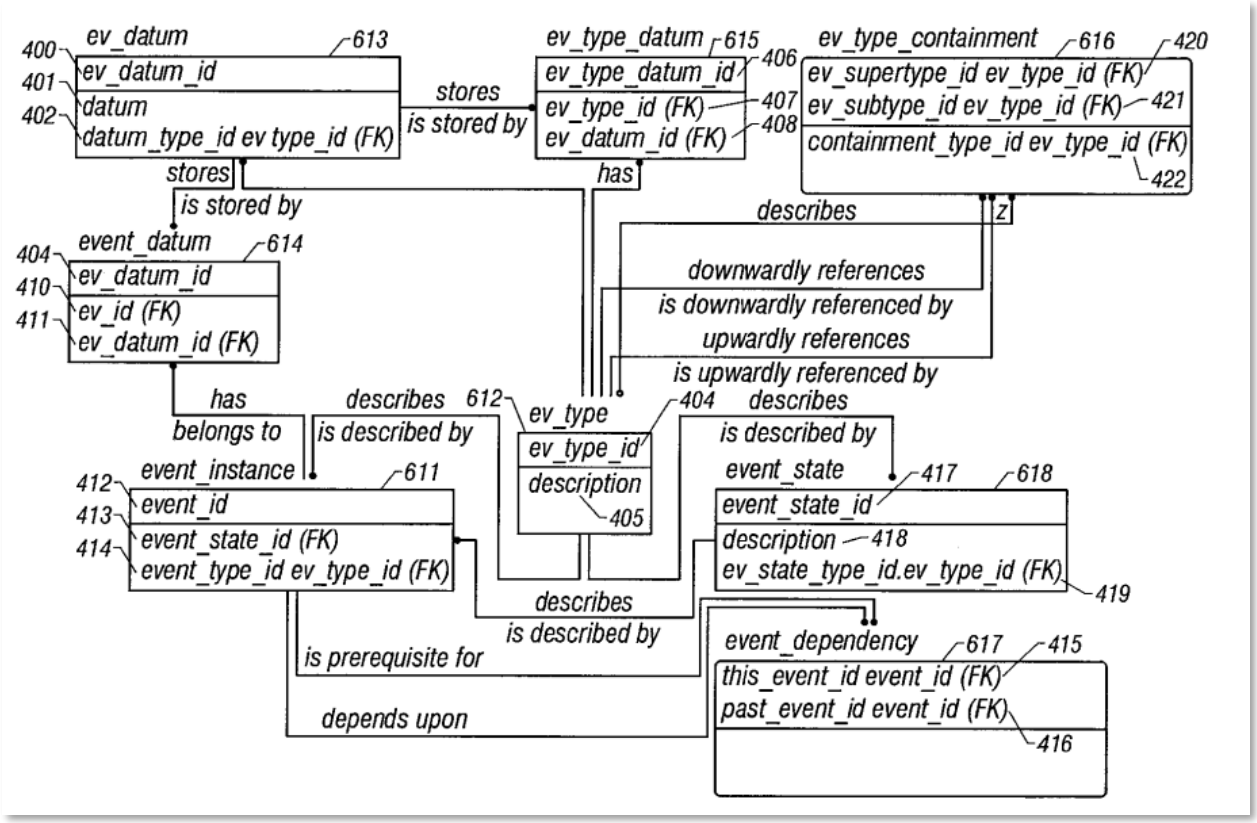
52. Moreover, the copies of Surjanto in the record are authentic. Exhibit 1008 is a complete copy of Surjanto that, according to Dr. Hall-Ellis, was obtained from the online catalog of the Linda Hall library and was available by 2000. Ex. 1010, ¶39. Based on my significant experience as a reader and 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)

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

54. 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.

55. 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)

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

57. 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 SURJANTO RENDERS OBVIOUS CLAIMS 1, 3-12, 14-23, 25-33

58. Ground 1 proposes the relational database structures of Althoff modified to include (1) Surjanto's XML document management extensions (*i.e.*, XCoP). 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.

2. A POSITA Would Have Been Motivation to Combine Althoff and Surjanto

59. Althoff 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. Surjanto specifically teaches a

database management system that improves content management of documents.

60. Like Althoff and Surjanto, the ’855 Patent is directed to the field of computer data storage, more specifically storing and managing data sets in a database system. Ex. 1001, 1:15-18. All four references, therefore, are directed to the same field of endeavor, and are analogous art.

61. In my opinion, a POSITA would have been motivated to combine Althoff with Surjanto by incorporating Surjanto’s XML document management extensions (*i.e.*, XCoP) with Althoff’s OO databases, 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.

62. A POSITA would expect several benefits from incorporating Surjanto’s teachings into Althoff, such as enabling “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 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 with Althoff's.

63. 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.

64. 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," which provides "a set of classes, objects, and relationships between classes." Ex. 1006, Abstract.

65. 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.

66. 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.

67. 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 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 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.

68. In my opinion, a POSITA would have combined Althoff and Surjanto with a reasonable expectation of success. Althoff's OO database and database model was compatible with Surjanto, as expressly taught by Surjanto. Indeed, Surjanto provides detailed instructions for implementing its extensions into a database like Althoff's. 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.

69. Moreover, the combination of Althoff and Surjanto would have yielded predictable results. A POSITA would have readily understood that combining Althoff 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. 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 to achieve the predictable benefits taught by Surjanto.

3. Claim 1

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

70. 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.

71. 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 hardware-based processor; and

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

73. In my opinion, a POSITA would understand that a general purpose computer includes a central processing unit, also known as a ***hardware-based processor***. The term "general purpose computer" was a well-known term of art that a POSTIA would understand to include several components, including a ***hardware-based 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 ***hardware-based processor***, as claimed.

(c) 1[b]: a data repository that contains the data sets and one or more layers of metadata of the data in the data sets, comprising:

74. In my opinion, Althoff discloses a data repository that contains the data sets and one or more layers of metadata of the data in the data sets, as claimed. Althoff discloses the ***data repository*** as a "relational database . . . implemented using a relational database engine." Ex. 1006, 6:42-55.

75. 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.

76. Althoff discloses a data repository ***that contains 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.

77. 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.

78. 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.

79. 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.

80. 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.

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

81. In my opinion, Althoff discloses a class module configured to define at least one class of the elements and store the class, as claimed.

82. Althoff discloses ***a class module***. As discussed below, the ***elements*** stored in the ***elements module*** are instanced objects of a class. *Infra*, Section XI3(; *see also* 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 elements*** (for example, a Manufacturer class of the Manufacturer objects).

83. 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.

84. 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.

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

85. In my opinion, 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.

86. 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 IXA. 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").

87. 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,* Section XI(d)-(e). Indeed, Figure 3 of Althoff teaches that a "Class Property" object "is an attribute of" a "Class" object. *Id.*

88. 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.

89. 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.

(f) 1[e]: 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;

90. In my opinion, 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.

91. 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.

92. 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]

93. Althoff's "relationships 103" is ***a class attribute module configured to define and store*** one or more class-attribute associations, as

[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.

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.

94. 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 report to one or more 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.

95. In my opinion, 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***.

(g) 1[f]: an element attribute module for the at least one class configured to: define and store the attributes of the elements in the class; and transmit the attributes and elements;

96. In my opinion, Althoff discloses an element attribute module for the at least one class configured to: define and store the attributes of the elements in the class; and transmit the attributes and elements, as claimed.

97. ***[A]n element attribute module for the at least one class configured to define and store the attributes of the elements in the class:*** In the Althoff-Surjanto combination, this limitation is disclosed by Althoff's user database model 230. Althoff discloses that "[t]he process of building and editing the user database model 230 ... simply comprises building and editing classes 101, ***objects*** [*i.e.* ***elements***]***, searchable properties 102*** [*i.e.* ***attributes***] and relationships 103 between classes 101, represented by rows, values and pointers in the meta-model." Ex. 1006, 7:67-8:5. As discussed above, searchable properties are attributes for a class of objects. *Supra*, Section XI(e)-(f). Moreover, as discussed below, the user database model 230 is used

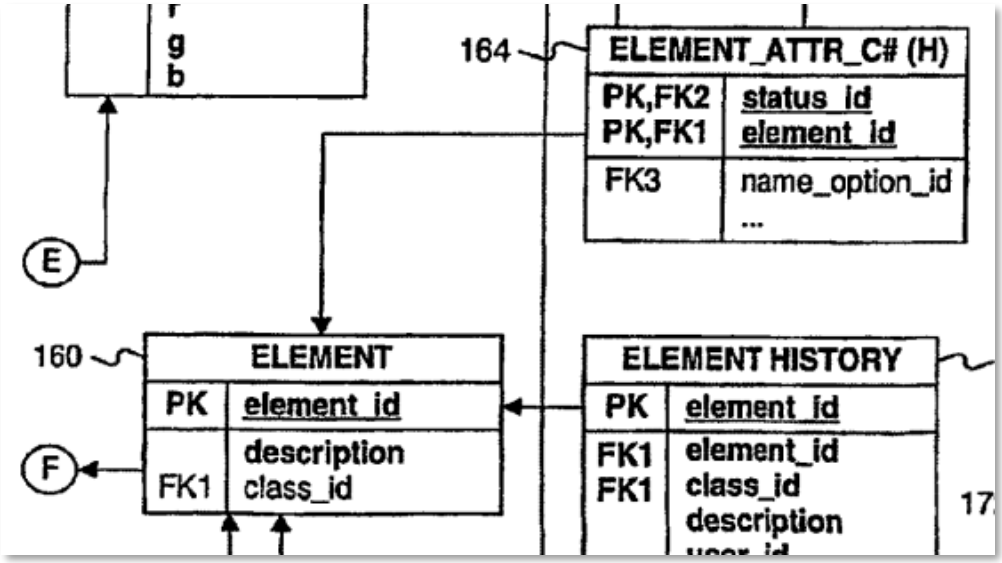
to provide objects and their searchable properties (*i.e.* ***attributes***) for the creation of class tables. Thus, Althoff discloses that the database model 230 defines and stores objects and their searchable properties (*i.e.* ***attributes of the elements in the class***).

98. ***[T]ransmit the attributes and elements:*** Althoff discloses that the user database model 230 transmits attributes and elements. Althoff teaches that "the system 200 translates the user model 230 into a user relational database 231" via a process described by Figure 7. Ex. 1006, 8:18-20. This is done "using a set of SQL commands 232 for specifying and creating relational structures in the user relational database 231" which "specify relational database tables, ***properties of those tables, columns within those tables***." *Id.*, 8:22-25. These SQL commands "are ***transmitted*** to the relational database engine 240" upon a "triggering event." *Id.*, 8:25-30. A POSITA would understand that the transmitted SQL command includes objects (*i.e.* ***elements***) and searchable properties (*i.e.* ***attributes***) because the SQL command specifies properties and columns of the tables built into the user relational database 231.

99. In particular, Figure 7, which depicts "[t]he process for translation between object-oriented and relational database structures" (Ex. 1006, 22:65-23:12), is described as "creat[ing] a table ... to correspond to"

each class in the data model (hereinafter called an "individual class table"). *Id*. Althoff teaches that "[w]hen an object [*i.e.* ***element***] 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." *Id.*, 23:19-25.. Althoff further teaches that the columns in the table correspond with each object's attributes, including "class identifier," "object type," and each "searchable property 102 for the class 101." *Id.*, 23:25-50. Thus, a POSITA would understand that Althoff describes that elements and their attributes populate each individual class table from the SQL command derived from user database model 230 (*i.e.* ***element attribute module***).

100. Also, POSITA would understand that the '855 Patent is consistent with this read of Althoff because Figure 1B of the '855 Patent uses a relational arrow to show that the elements and attributes are ***transmitted from the element attribute module to the element module***, where both the element attribute module and element module are within the relational database. The '855 patent does not otherwise provide any explanation for how elements and attributes are transmitted from the element attribute module to the element module. This disclosure is similar to what is described in Althoff (indeed, Althoff expressly discloses that the table is populated using transmitted SQL commands).



Ex. 1001, Fig 1B.

(h) 1[g]: an element module configured to receive the transmitted attributes and elements from the element attribute module, store the attributes and elements, and uniquely identify the elements, wherein the elements are identified by unique element identifications;

101. In my opinion, Althoff discloses an element module configured to receive the transmitted attributes and elements from the element attribute module, store the attributes and elements, and uniquely identify the elements, wherein the elements are identified by unique element identifications, as claimed.

102. ***[A]n element module configured to receive the transmitted attributes and elements from the element attribute module:*** In the Althoff-Surjanto combination, this limitation is disclosed by Althoff's individual class

tables. 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," *i.e.* an individual class table. *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. As explained above, Althoff teaches that SQL commands containing attributes and elements from the user database model 230 are transmitted to the relational database 202 to create these individual class tables.

103. ***[A]n element module configured to store the attributes and elements:*** Althoff 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. Thus, a Althoff's individual class tables are ***element modules*** because each row of the table stores objects (*i.e.*, ***elements***).

104. Althoff further teaches that the individual class tables are configured to store those objects' searchable properties (*i.e.* ***attributes***), as well as other object attributes. Althoff teaches that each individual class table

has columns for attributes like each object's "class identifier," "object type" and "searchable property" (which as discussed above in limitation 1[d] is an ***attribute***). Ex. 1006, 23:25-50.

105. Figure 8 depicts an example of these individual class tables. Althoff describes that in the Figure 8 example, "table 800" is a table for the Component class and has columns for "name of the component," "size of the component," "name of the manufacturer" of the component, "[the manufacturer's] city" and "country." Ex. 1006, 24:18-25.

106. ***[A]n element module configured to uniquely identify the elements, wherein the elements are identified by unique element identifications:*** 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***).

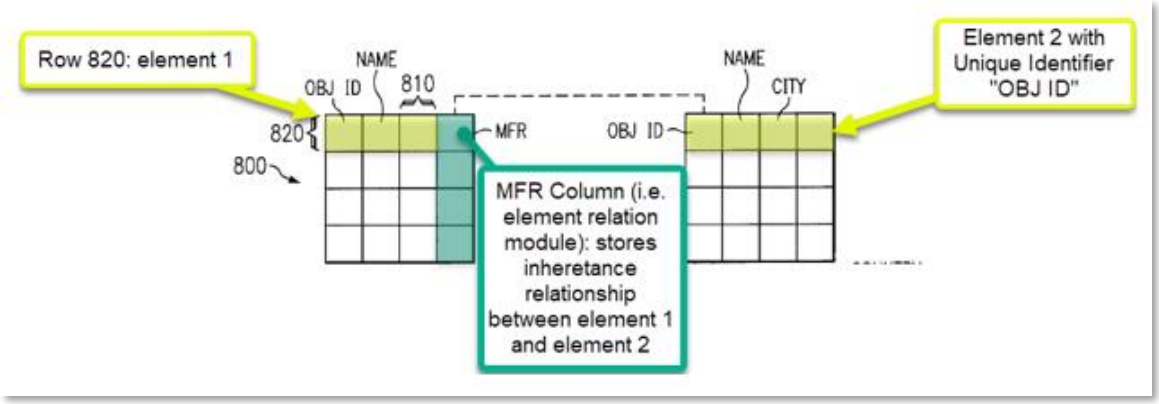
(i) 1[h]: an element relation module configured to store one or more relationships between the elements in the element module;

107. In my opinion, 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 XI3(d)-(h). Thus, in my opinion, column 810 (*i.e.*, the ***element relationship module***) stores inheritance relationships between objects.

108. Althoff discloses that Figures 7 and 8 are examples 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.").

109. 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.

110. 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.

111. 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.

112. 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 between the elements in*** tables—including ***the element module*** which is defined in Althoff as a specific implementation of a table (*i.e.*, the class tables).

(j) 1[i]: an element history module configured to store one or more indicators of a modification to the elements; and

113. In my opinion, 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.*

114. 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.

115. 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.

116. 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 XI2. This would enable a specific implementation of version control which was a well-known technique at the time of the ’855 Patent. *Id*.

(k) 1[j]: an element document module configured to store formatting data for grouping a subset of the elements.

117. In my opinion, Althoff discloses an element document module configured to store formatting data for grouping 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 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.

118. In my opinion, a POSITA would understand the data such as “maximum display width,” “Custom Label” and the “relative order” for displaying the rows and columns of the elements to be formatting as claimed.

4. Claim 3: The computerized system of claim 1, wherein the data repository further comprises:

(a) 3[a]: 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; and

119. In my opinion, 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 XI3(d). 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.

120. 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 Section XI3(e)-(f). 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.

121. 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.

122. 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).

123. 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.

124. 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.

125. 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 XI3(e)-(f). 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.

126. Finally, Althoff's class Pointer Valid Class ***defines and stores the relationships between the elements,*** as claimed. As noted below, Althoff

discloses elements as objects (i.e. rows) in the class tables. *Infra,* Section Section XI3(f)-(h).

127. 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* Section XI3(g).

128. 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.

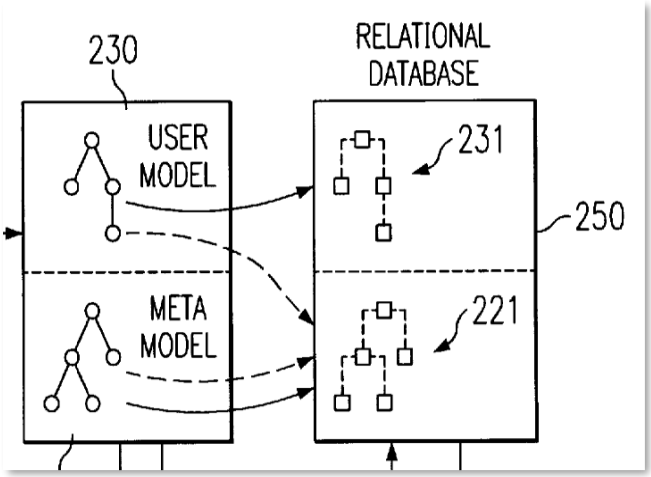
(b) 3[b]: a list option module configured to enumerate one or more picklist attributes of a picklist type.

129. Althoff's Enumerated Valid Value and Range Valid Value each disclose ***a list option module configured to enumerate one or more picklist attributes of a 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.*

130. In my opinion, a POSITA would understand that by restricting the value of the Property Data Type to a predefined set of choices Althoff's Enumerated Valid Value or Range Valid Value enumerate one or more picklist attributes of a picklist type.

131. 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.

5. Claim 4: The computerized system of claim 1, wherein the data repository further comprises at least one of a named text module configured to store textual representations of query, control, and display mechanisms that relate to the elements.

132. In my opinion, 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.

133. 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.

134. 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.

135. 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.

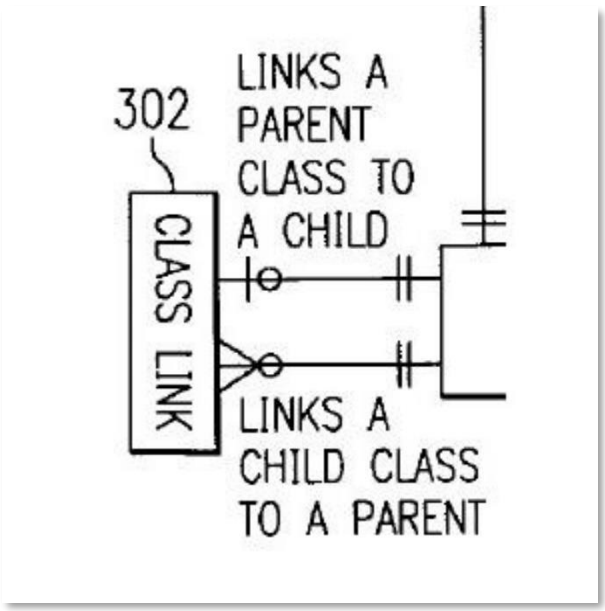
136. Further, in my opinion a POSITA would understand that the combination of queries, access control mechanisms, and display mechanisms are ***stored in*** Althoff's query model (*i.e.*, ***the named text module***) because Althoff discloses "display properties" belonging to "the classes which were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

6. Claim 5: The computerized system of claim 1, wherein the data repository further comprises: a tuple module configured to generate and store one or more tuples, wherein the tuples associate subsets of the elements in accordance with types associated with the subsets of the elements.

137. 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.

138. 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.

139. 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 Section XI3(d). 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.

140. 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***.

7. Claim 6: The computerized system of claim 5, wherein the tuple module uses recursion to link a first folder to at least one additional folder in a hierarchy.

141. In my opinion, Surjanto's ***tuple module uses recursion to link a first folder to at least one additional folder in a hierarchy,*** as claimed. Surjanto teaches that "a table ***hierarchy*** can be specified" based on a "defined type hierarchy" established for subtypes corresponding to sub-classes of elements. Ex. 1008, 13.

142. In my opinion, A POSITA would understand that Althoff's table hierarchy ***links folders that contain element based on the type hierarchy*** defined for each sub-class of elements.

143. Further, in my opinion, a POSITA would understand that recursion is a fundamental concept in computer programming that has existed for decades, and would have been motivated to apply ***recursion*** in this context ***to link a first folder containing sub-classes of elements to at least one***

additional folder in the hierarchy. Indeed, Surjanto teaches the use of recursion in a similar context for creating object child fragments and their corresponding aggregation relationships: "[i]n case of the creation of a (DTD) fragment object, fragmentation markups within the textual contents of the fragment are recognized and the createObject operation is ***recursively performed for each child fragment***." Ex. 1008, 9. In other words, Surjanto's tuple module links a first folder containing a sub-class of the elements to at least one additional folder in the hierarchy. A POSITA would understand that the fundamental programming concept of recursion would be particularly well-suited to linking folders contained in the hierarchy because folders in a hierarchy are nested within each other.

144. A POSITA would have been motivated to combine Surjanto's tuple module and Althoff's tuple module to place objects (*i.e.*, elements) associated with parent and child classes in linked folders.. Indeed, Althoff discloses "many-to-many parent/child class relationships 103" that are "cross-link[ed]" which would be well suited to organization in a ***hierarchy of folders, where recursion would be used to link those folders***. Ex. 1006, 12:29-34.

8. Claim 7: The computerized system of claim 6, wherein the tuple module includes one or more entries, and the data repository filters the one or more entries into one or more folders.

145. In my opinion , Surjanto discloses the tuple module ***wherein the tuple module includes one or more entries, and the data repository filters the one or more entries into one or more folders,*** as claimed. In the Althoff-Surjanto combination, this limitation is disclosed by Surjanto. Surjanto teaches a "supertype" for each subtype that corresponds to a sub-class. Ex. 1008, 16.

146. In my opinion, a POSITA would understand that a "supertype" ***includes one or more entries*** corresponding to sub-classes of elements belonging to each subtype defined for Surjanto's table hierarchy (*i.e.*, ***the tuple module***). Moreover, a POSITA would further understand that Surjanto's ***data repository*** teaches ***filtering the one or more entries into one or more folders.*** The supertype taught by Surjanto encompasses a hierarchy of folders for various sub-classes of elements that are organized by subtypes in Surjanto's table hierarchy.

As described above, a POSITA would have been motivated to combine Surjanto's tuple module and Althoff's tuple module to place objects (*i.e.*, elements) associated with parent and child classes in linked folders. *Supra*, Section XI2. A POSITA would have also understood that Surjanto's

disclosure of filtering subtypes belonging to a supertype into one or more folders could readily be applied to ***filter*** the objects (*i.e.*, ***entries***) belonging to Althoff's parent and child classes ***into one or more linked folders***. *Id.*

9. Claim 8: The computerized system of claim 1, wherein the class attribute associations are identified by logical names and class names.

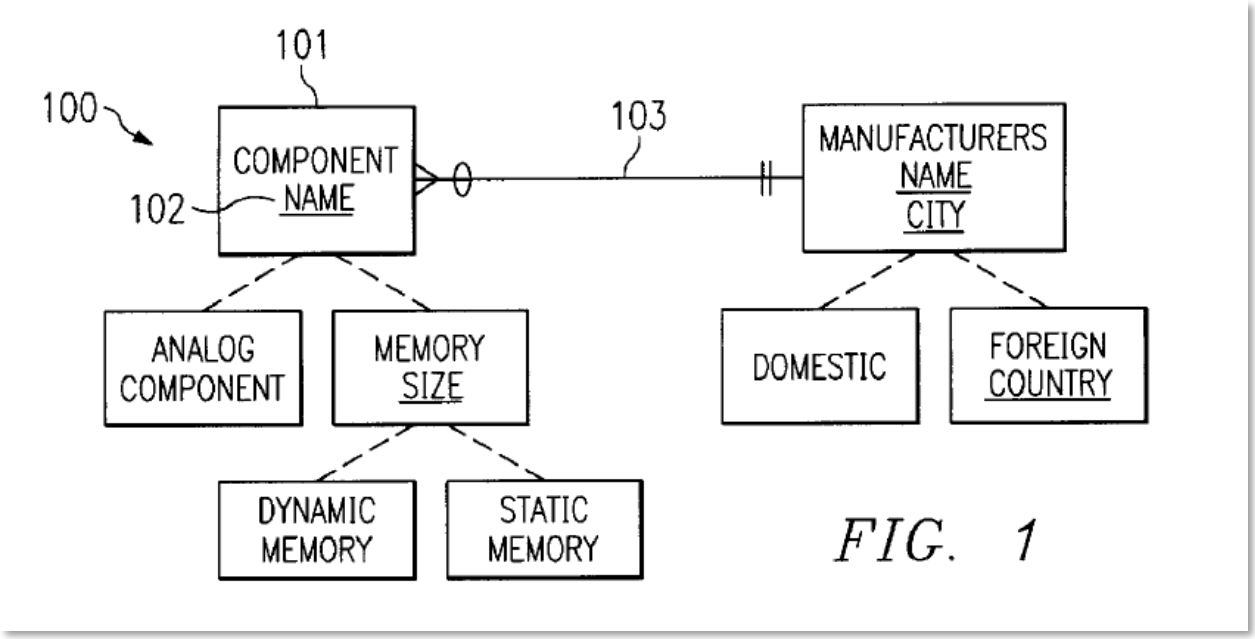
147. Althoff discloses ***wherein the class attribute associations are identified by logical names and class names***, as claimed. Althoff's Class Property objects and Pointer Class Property objects are the attributes, and Althoff's class Class objects are the classes. *Supra*, Section XI3(d)-(e). Each 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.

148. 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

Section XI3(d)-(j). In software development, logical name is 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.

10. Claim 9: The computerized system of claim 1, wherein the element attribute module is identified using a name of the class.

149. In my opinion, 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 XI3(f)-(h). 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.



Ex. 1006, Fig. 1.

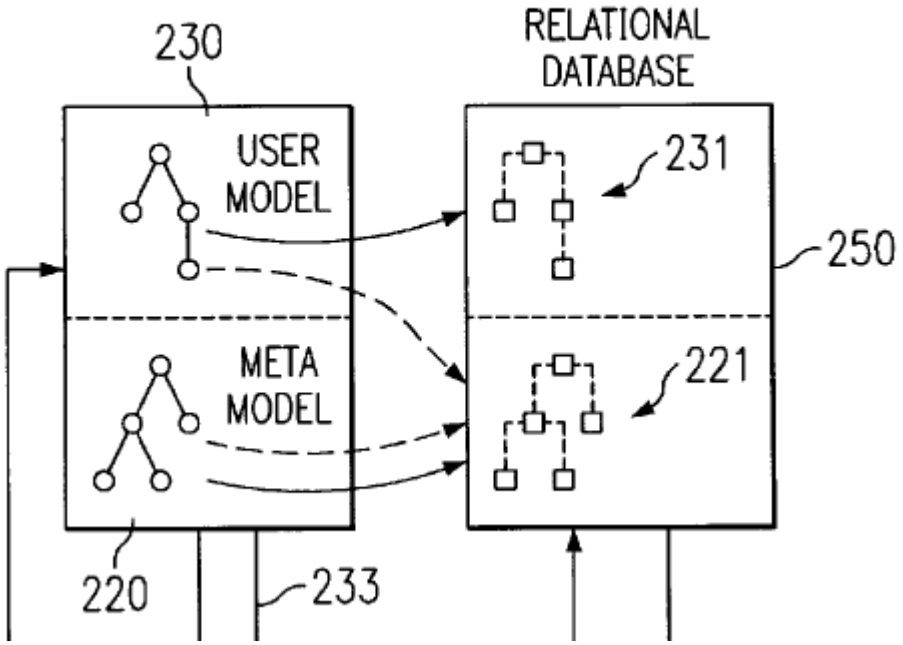
150. Each class is represented by a relational database class table of the same ***name of the classes***: "table 800 component, a table 800 analog component, a table 800 memory, a table 800 dynamic memory, a table 800 static memory, a table 800 manufacturer, a table 800 domestic manufacturer, and a 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***.

11. Claim 10: The computerized system of claim 1, wherein the processor is configured to generate a data model of the elements based on the layers of metadata.

151. In my opinion, Althoff's meta-model discloses ***a processor configured to generate a data model of the elements based on the 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 XI3(b)-(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.

152. 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 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

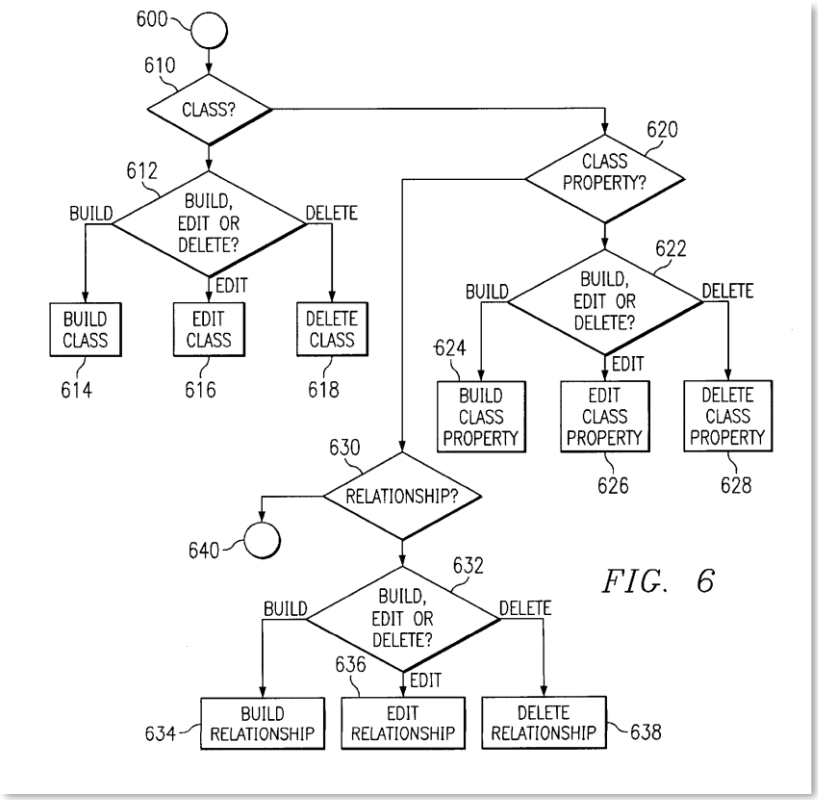
12. Claim 11: The computerized system of claim 1, wherein the processor is configured to generate a dynamic user interface for manipulating the elements based on the layers of metadata stored in the data repository.

153. In my opinion, Althoff's discloses generating a dynamic user-interface for accessing and manipulating the elements based on 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 XI3(b). 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.

154. 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.

155. In my opinion, a POSITA would understand that Althoff's GUI is ***based on 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 12, 14-23, 25-33

156. In my opinion, the limitations of claims 12, 14-23, and 25-33 are rendered obvious by Althoff and Surjanto for the same reasons as discussed in connection with claims 1 and 3-11, as set forth in the following chart. *Supra* Section XI1-12.

Claim Limitation	Corresponding Claim Limitation(s)
12[pre]	1[pre]
12[a]	1[b], 1[c]
12[b]	1[d]
12[c]	1[e]
12[d], 12[e]	1[f]
12[f], 12[g]	1[g]
12[h]	1[h]
12[i]	1[i]
12[j]	1[j]
14[a]	3[a]
14[b]	3[b]
15	4
16	5
17	6
18	7
19	8
20	9
21	10
22	11
23[pre]	1[pre], 1[a]
23[a]	1[b], 1[c]

Claim Limitation	Corresponding Claim Limitation(s)
23[b]	1[d]
23[c], 23[d]	1[e]
23[e], 23[f]	1[f]
23[g], 23[h]	1[g]
23[i]	1[h]
23[j]	1[i]
23[k]	1[j]
25	3[pre]
25[a]	3[a]
25[b]	3[b]
26	4
27	5
28	6
29	7
30	8
31	9
32	10
33	11

Claim Limitation	Corresponding Claim Limitation(s)

XII. GROUND 2: ALTHOFF IN VIEW OF SURJANTO IN FURTHER VIEW OF NYE RENDERS OBVIOUS CLAIMS 2, 13, 24

157. Ground 2 further modifies the Althoff-Surjanto combination with Nye's event model. The resulting combination renders obvious claims 2, 13, and 24.

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

158. Althoff, Surjanto 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. Surjanto specifically teaches a database management system that improves content management of documents. 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.*

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

160. Like Althoff, Surjanto, and Nye, the '855 Patent is directed to the field of computer data storage, more specifically storing and managing data sets in a database system. Ex. 1001, 1:15-18. All four references, therefore, are directed to the same field of endeavor, and are analogous art.

161. In my opinion, a POSITA would have been motivated to combine Althoff, improved by Surjanto's document management extensions, with 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 the combination of Althoff and Surjanto. 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.

162. 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.

163. 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, improved by Surjanto's document management extensions, for several reasons.

164. 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.

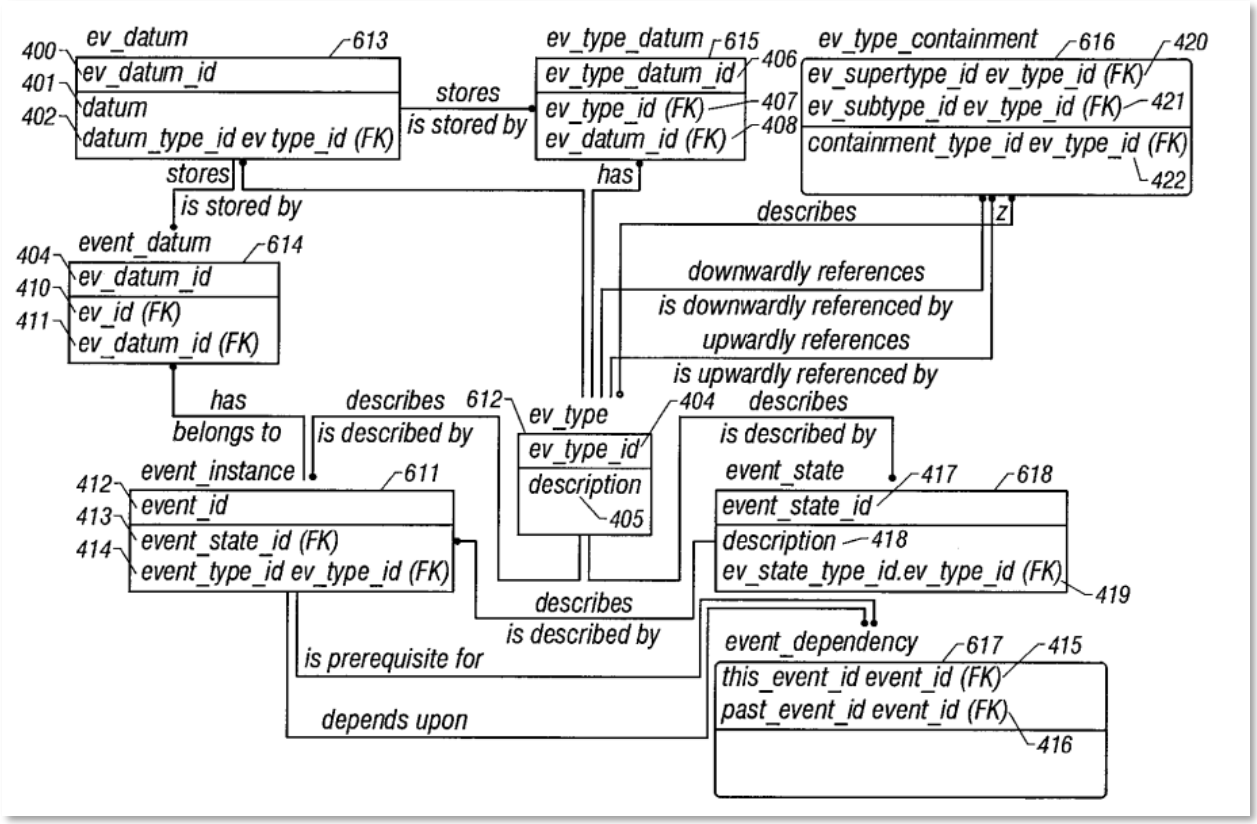
165. Second, a POSITA would know that integrating Nye's teaching of a state machine into Althoff's relational database, improved by Surjanto's document management extensions, would further Althoff's state goal of representing the OO model in a relational model. Althoff taught "it would be advantageous 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.*

166. As such, as POSITA would be motivated to combine Nye's event model with Althoff, improved by Surjanto's document management extensions, to achieve a set of relational database structures to while maintaining the traditional advantages of the OO model.

167. A POSITA would have known this combination of Althoff, improved by Surjanto's document management extensions, 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 and Surjanto's document management extensions, 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 combined with by Surjanto's document management extensions would have been expected to succeed. *Id.*



Ex. 1005, Fig. 4.

168. A POSITA would have found it obvious to incorporate Nye's teaching of event models to Althoff's teaching of an OO database with

Surjanto's document management extensions 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.

169. A POSITA would have combined the teachings of Althoff with Surjanto's document management extensions 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 '855 Patent.

170. Moreover, the combination of Althoff with Surjanto's document management extensions and Nye would have yielded predictable results. A POSITA would have readily understood that combining Althoff improved by Surjanto's document management extensions and Nye would enable event

modeling in Althoff's database, and would thus further the functionality of the combination of Althoff's system with Surjanto's document management extensions. 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 environments taught by Althoff and Surjanto.

2. Claim 2: The computerized system of claim 1, wherein the data repository further comprises:

(a) 2[a]: a state machine module including a state machine transition module and a status module configured to define and store a state machine type,

171. In my opinion, Nye discloses ***a state machine module including a state machine transition module and a status module configured to define and store a state machine type,*** as claimed.

172. 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, in my opinion Nye's event model is a ***state machine module*** that represents the various event states and the transitions between those states.

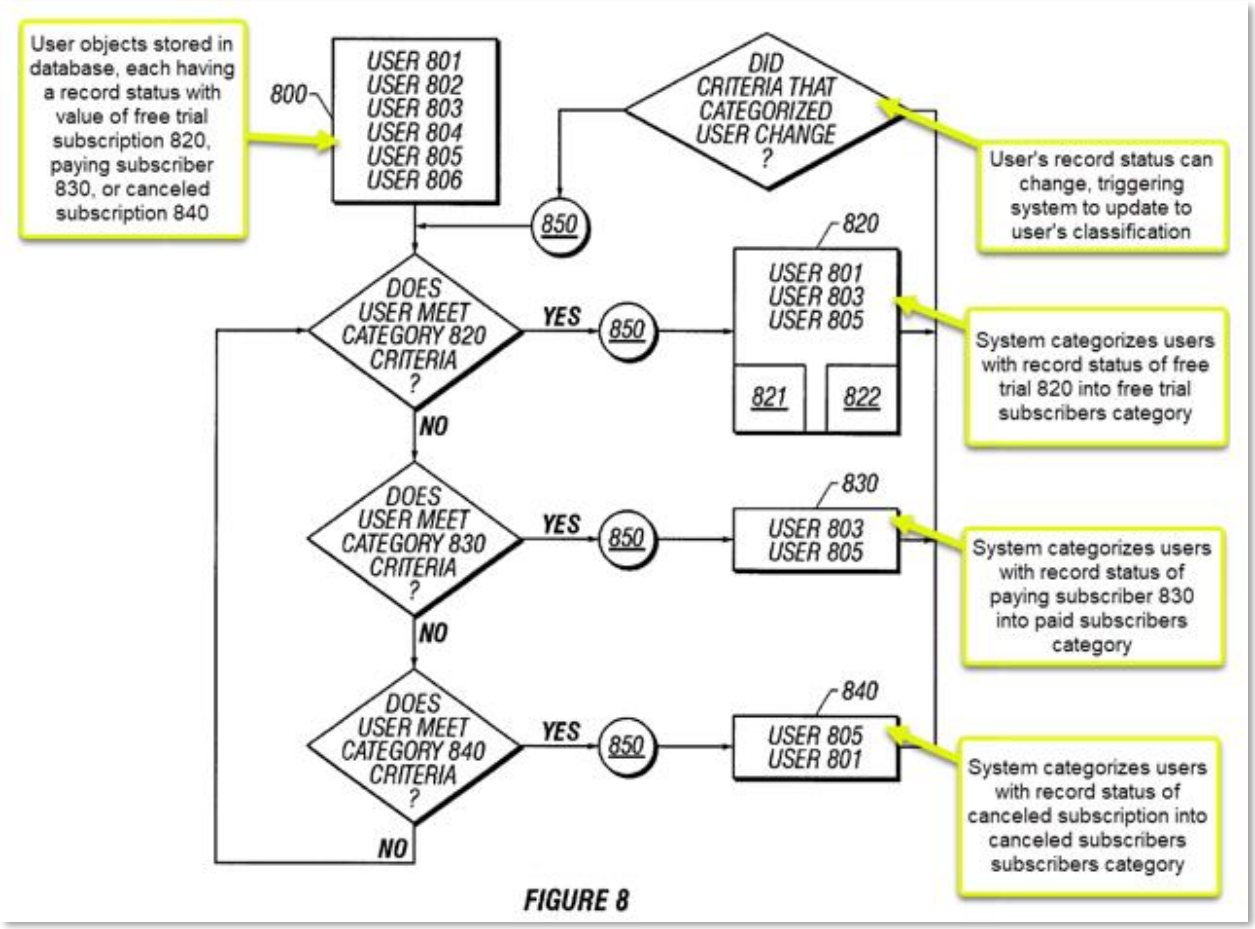
173. Moreover, in my opinion Nye's state transitions disclose ***a state machine transition module*** included in the event module (i.e. state machine module). Nye teaches that the event model "***provides a way to flexibly handle state transitions***." *Id.*, 5:28-42. Nye teaches that an "event, is 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. In my opinion, Nye also discloses ***defining and storing 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.'" Ex. 1005, Abstract.

174. In my opinion, Nye's event module further discloses ***a state machine module including a status module configured to define and store a state machine type:*** As noted above, Nye's event model teaches ***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 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.*, ¶121; Ex. 1005, 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***." Ex. 1005, 7:10-13.

175. 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: "[i]n 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

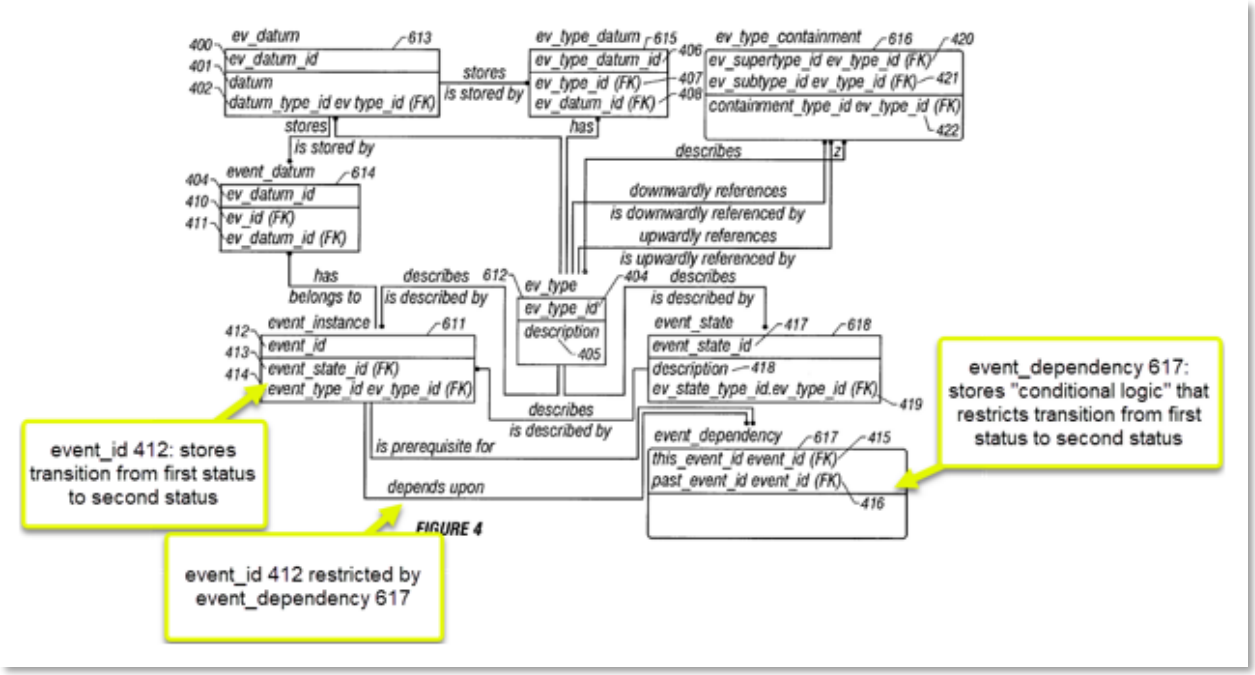
database of users 800 can be categorized according to the users 801-806 current status." *Id.*, 7:23-26. As shown below in Figure 8, Nye teaches a ***state machine*** comprised of ***three event states for a user's record status***: "[f]ree trial subscription category 820 . . . paying subscribers category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



Ex. 1005, Fig. 8 (annotated).

(b) 2[b]: the state machine type storing a first status of one element and a transition to a second status of the one element.

176. In my opinion, Nye discloses ***the state machine type storing a first status of one element and a transition to a second status of the one element,*** as claimed. As noted above, 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." Ex. 1005, 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***: "[e]vent 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 of a ***state machine type*** 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 (annotated).

3. Claims 13 and 24

177. In my opinion, claims 13 and 24 mirror claim 2, as set forth in the following chart. Moreover, in my opinion, the combination of Althoff, Surjanto, and Nye renders these claims obvious for the same reasons as set forth above for claim 2. *Supra* Section XII1-2.

Claim Limitation	Corresponding Claim Limitation(s)
13[pre]	2[pre]
13[a]	2[b]
13[b]	2[a]
24[pre]	2[pre]
24[a]	2[b]

Claim Limitation	Corresponding Claim Limitation(s)
24[b]	2[a]

XIII. GROUND 3: ALTHOFF IN VIEW OF SURJANTO IN FURTHER VIEW OF MOORE RENDERS OBVIOUS CLAIMS 4, 6-7, 15, 17-18, 26, 28-29

178. In my opinion, as shown herein, the combination of Althoff, Surjanto, and Moore renders obvious claims 4, 6-7, 15, 17-18, 26, and 28-29. While these claims are all covered in Ground 1 (the combination of Althoff and Surjanto), Patent Owner may argue that the combination of Althoff and Surjanto does not disclose the "named text module" and "tuple module" that is the subject of these claims. While such an argument would be wrong for the reasons set forth in Ground 1, these limitations are clearly disclosed by Moore.

1. Motivation to Combine Althoff, Surjanto, and Moore

179. Althoff, Surjanto, 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. Surjanto specifically teaches database extensions called "XCoP (XML Content Repository," which "manages textual XML document contents together with their structural information." Ex. 1008, 9. 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.

180. Like Althoff, Surjanto, and Moore, the '855 Patent is directed to the field of computer data storage, more specifically storing and managing data sets in a database system. Ex. 1001, 1:15-18. In my opinion, all four references, therefore, are directed to the same field of endeavor, and are analogous art.

181. In my opinion, a POSITA would have been motivated to combine Althoff and Surjanto with Moore by incorporating Moore's disclosures of virtual folders of database items with Althoff's relational databases and Surjanto's teachings of XML document management extensions. 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 Surjanto). 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.

182. 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.

183. 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)."

184. 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 Surjanto's teachings of XML document management extensions expand the types of files that can be stored to include XML documents. 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.

185. 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 Surjanto.

186. In my opinion, a POSITA would have combined Althoff, Surjanto, 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 Surjanto. 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.

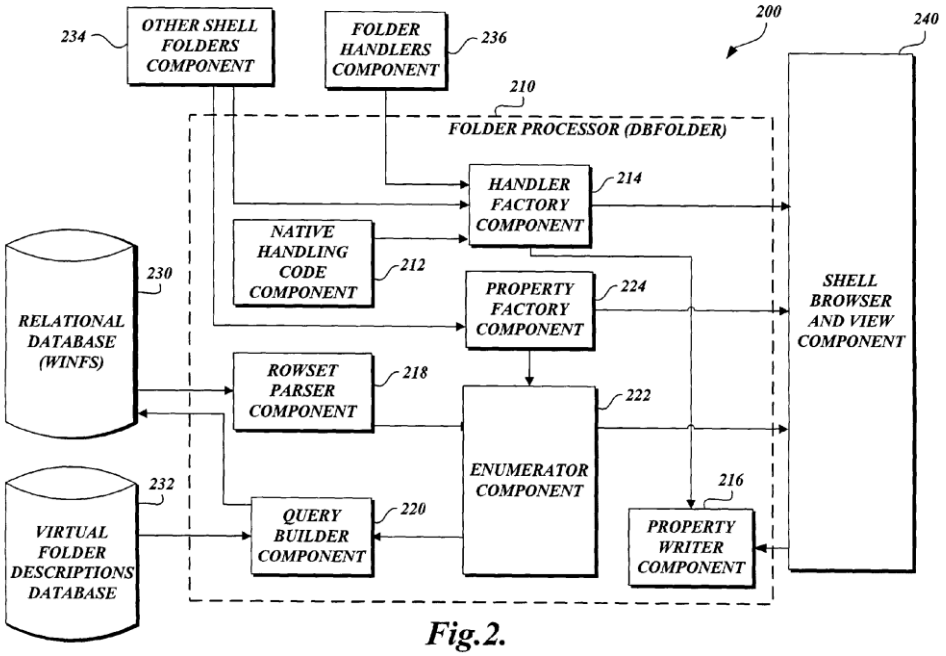


Fig.2.

Ex. 1007 at Fig. 2. The implementation of Moore's virtual folder system into the combination of Althoff and Surjanto was well within the capabilities of a POSITA at the time of the '855 Patent.

187. A POSITA would understand that Moore was compatible Althoff and Surjanto'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.

188. Moreover, the combination of Althoff, Surjanto, and Moore would have yielded predictable results. In my opinion, a POSITA would have readily understood that combining Althoff and Surjanto with Moore would result in a database system and database structures as taught by Althoff and Surjanto 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 Surjanto, to achieve the predictable benefits taught by Moore.

2. Claim 4: The computerized system of claim 1, wherein the data repository further comprises at least one of a named text module configured to store textual representations of query, control, and display mechanisms that relate to the elements.

189. 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.

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

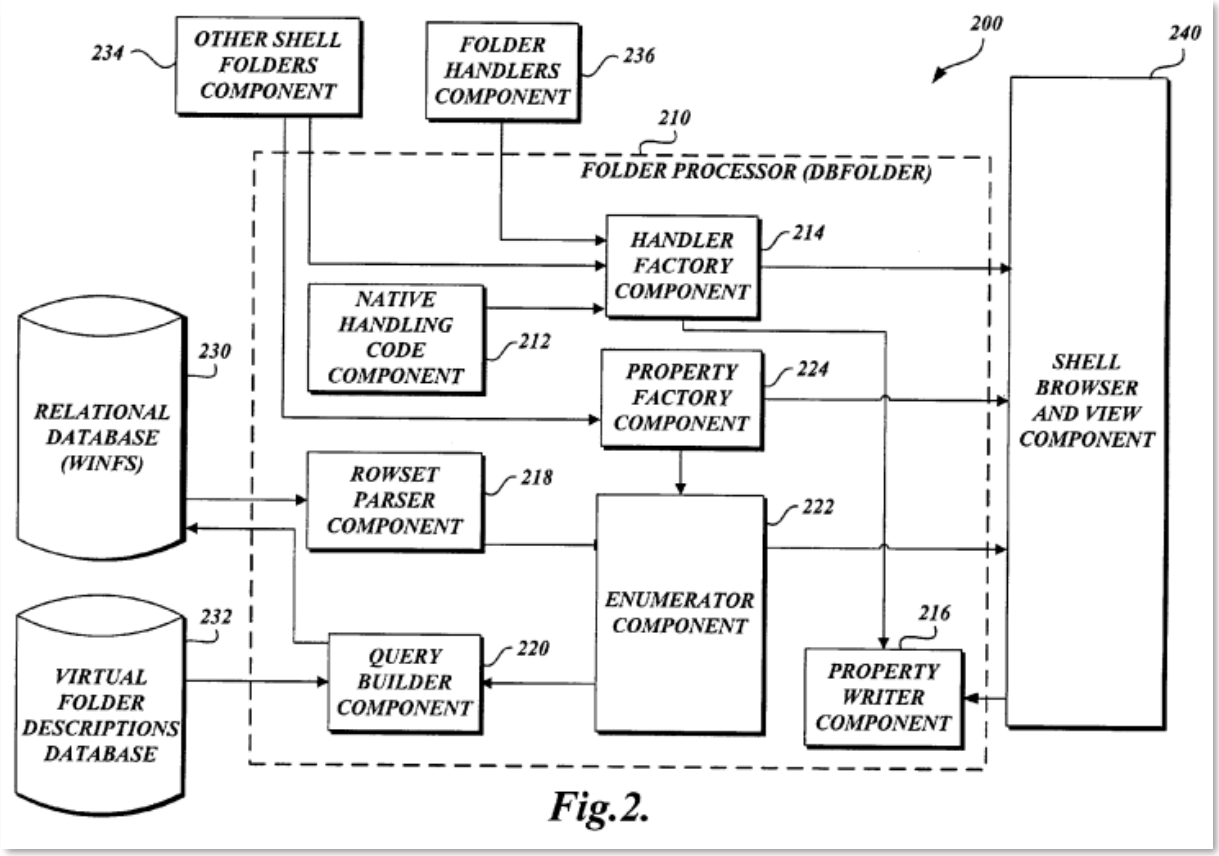


Fig.2.

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.

191. 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.

192. 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.

193. 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.

194. 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.

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

a string (*i.e.*, ***textual representation***) query for filtering data (*i.e.* ***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).

196. 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 elements***), the "virtual folder descriptions" is a ***textual representation*** of those mechanisms.

197. 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

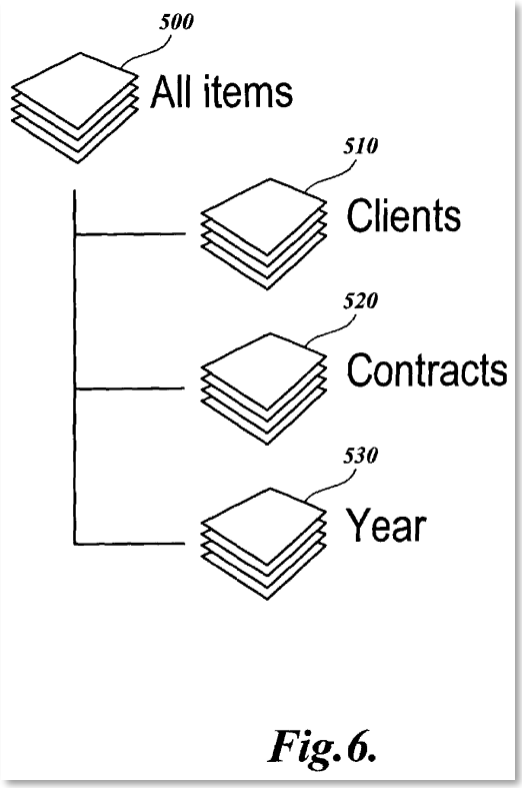
database of file and non-file items is much like Althoff's database of objects, containing ***elements*** (*i.e.*, ***file and non-file items***) and ***attributes*** about those elements (*i.e.*, ***item properties***), and Surjanto's database management system for document fragments (*i.e.*, ***elements***). 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 6: The computerized system of claim 5, wherein the tuple module uses recursion to link a first folder to at least one additional folder in a hierarchy.

198. In my opinion, Moore discloses the computerized system of claim 5, wherein the tuple module uses recursion to link a first folder to at least one additional folder in a hierarchy, as claimed.

199. ***[W]herein the tuple module link[s] a first folder to at least one additional folder in a hierarchy:*** 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, 8:23-28. Thus, in my opinion, the virtual folder descriptions database ***links*** the folder descriptions (*i.e.*, ***a first folder to at least one additional folder***).

200. Moore teaches that linked folders are contained ***in a hierarchy***, where virtual folders can "further include[] virtual folders." Ex. 1007, 11:62-65. For example, as depicted in Figure 6 (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.*, 11:55-59. 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.*, 11:62-65. Moore depicts and describes several over examples of hierarchical virtual folders in Figures 6-9.



Ex. 1007, Fig. 6.

201. Moore also teaches that "[t]he virtual folders expose 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. In my opinion, a POSITA would understand that since the virtual folder can itself contain folders (and folders can further contain more folders), that results in ***folders linked in a hierarchy***.

202. 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." Ex. 1007, 7:59-63.

203. ***[W]herein the tuple module uses recursion to link:*** In my opinion, a POSITA would understand recursion as a term of art in the field of software development. Recursion is a method of design where you defining a problem, or the solution to a problem, in terms of a simpler version of itself. Recursion allows developers to break apart a problem into smaller repeatable subtasks. Recursion is a fundamental concept in computer programming that has existed for decades. In my opinion, a POSITA would have been motivated to apply ***recursion in this context to link a first folder to at least one additional folder in the in the virtual folder hierarchy***. A POSITA would understand that the recursion would be particularly well-suited to linking

folders contained in the hierarchy because folders in a hierarchy are nested within each other. Recursion is an especially well-suited way to model a nested system, such as this type of folder hierarchy.

204. Indeed, as described above, Surjanto teaches the use of recursion in a similar context for creating object child fragments and their corresponding aggregation relationships. To the extent that recursion is not already disclosed by Moore's nested hierarchy of virtual folders, a POSITA would have been motivated to apply this teaching from Surjanto to create the nested virtual ***folder hierarchy*** disclosed by Moore, because Surjanto similarly discloses using ***recursion to link*** related fragments (*i.e.*, objects or elements).

4. Claim 7: The computerized system of claim 6, wherein the tuple module includes one or more entries, and the data repository filters the one or more entries into one or more folders.

205. In my opinion, Moore discloses the tuple module includes one or more entries, and the data repository filters the one or more entries into one or more folders, as claimed.

206. ***[W]herein the tuple module includes one or more entries, and the data repository filters the one or more entries into one or more folders:*** As explained above, 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, 8:23-28. Thus, in my opinion, the virtual folder descriptions database links ***one or more entries*** to the folder descriptions (*i.e.*, ***folders***) via the initial filter (*i.e.*, ***the data repository filters***).

207. Moreover, in my opinion Moore also teaches that "the virtual folders expose 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." Ex. 1007, 9:63-66. Since the virtual folder can itself contain folders (and folders can further contain more folders), that results in a hierarchy of ***one or more data entries*** contained in each folder depending on the specific view selected for the virtual folder system (since the folders are items in the relational database). For example, as described above, the virtual folder system ***filters the one or more entries into one or more folders*** so that "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." Ex. 1007, 22:2-6.

5. Claims 15, 17-18, 26, 28-29

208. In my opinion, the limitations of claims 15, 17-18, 26, and 28-29 are rendered obvious by Althoff, Surjanto, and Moore for the same reasons as discussed in connection with claims 4 and 6-7, as set forth in the following chart. *Supra*, Section XII1-4.

Claim Limitation	Corresponding Claim Limitation(s)
15	4
17	6
18	7
26	4
28	6
29	7

XIV. SECONDARY CONSIDERATIONS

209. 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

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

211. 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.

212. 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: August 28, 2023

Jules White, Ph.D.