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

_____

CORRECTED PETITION FOR *INTER PARTES* REVIEW OF U.S. PATENT NO. 8,224,855

TABLE OF CONTENTS

I. Standing 1

II. Identification of Challenge And Relief Requested 1

III. The ’855 Patent 1

A. Overview 1

B. Prosecution History of the ’855 Patent 2

IV. Level Of Ordinary Skill In The Art 2

V. Claim Construction 3

A. “module” 3

VI. Overview Of The Prior Art 4

A. Althoff (Ex. 1006) 4

B. Surjanto (Ex. 1008) 4

C. Nye (Ex. 1005) 7

D. Moore (Ex. 1007) 8

VII. Detailed Explanation of Invalidity Grounds 8

A. Ground 1: Althoff in view of Surjanto renders obvious claims 1, 3-12, 14-23, 25-33 8

1. Motivation to Combine 8

2. Claim 1 12

3. Claim 3: The computerized system of claim 1, wherein the data repository further comprises: 29

4. 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.....33

5. 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. 35

6. 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.....38

7. 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. 39

8. Claim 8: The computerized system of claim 1, wherein the class attribute associations are identified by logical names and class names.....40

9. Claim 9: The computerized system of claim 1, wherein the element attribute module is identified using a name of the class. 41

10. 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. 43

11. 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. 43

12. Claims 12, 14-23, 25-33.....45

B. Ground 2: Althoff in view of Surjanto in further view of Nye renders obvious claims 2, 13, 24.....47

1. Motivation to Combine 48

2. Claim 2: The computerized system of claim 1, wherein the data repository further comprises: 51

3. Claims 13, 24 56

C. Ground 3: Althoff in view of Surjanto in further view of Moore renders obvious claims 4, 6-7, 15, 17-18, 26, 28-29.....56

1. Motivation to Combine 56

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.....60

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.....64

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. 66

5. Claims 15, 17-18, 26, 28-29.....68

VIII. The Board Should Not Exercise Its Discretion to Deny the Petition.....68

IX. Mandatory Notices And Fees 69

A. Real Party-In-Interest 69

B. Related Matters.....69

C. Counsel and Service Information 69

D. Payment of Fees 70

X. Conclusion 71

EXHIBIT LIST

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.*, June 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.), ECF No. 33 ("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.

ServiceNow, Inc. ("ServiceNow" or "Petitioner") respectfully requests *inter partes* review of claims 1–33 (the "Challenged Claims") of U.S. Patent No. 8,224,855 (the "'855 Patent") pursuant to 35 U.S.C. §311 and 37 C.F.R. §42.100.

I. STANDING

Petitioner certifies that the '855 Patent is available for IPR and that Petitioner is not barred or estopped from presenting the grounds identified in this Petition.

II. IDENTIFICATION OF CHALLENGE AND RELIEF REQUESTED

Petitioner requests cancellation of claims 1–33 on the following grounds:

Ground	Reference(s)	Basis	Claims
1	Althoff, 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

III. THE '855 PATENT

A. Overview

U.S. Patent No. 8,224,855 (the "'855 Patent") is directed to object-oriented ("OO") data modeling, known in application software development since the '80s. Ex. 1003, ¶¶32-44. OO data models abstract engineering by abstracting interactions using classes of objects. *Id.*, ¶33. These developments began in the 1980s. *Id.*, ¶34. The '585 patent discloses basic OO concepts, *e.g.* objects, classes, attributes, and relationships. *Id.*, ¶¶35-37. OO model databases since the early 1990s allowed

developers to directly store classes, objects, attributes, relationships, etc. in the underlying databases. *Id.*, ¶40.

The '855 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.*, 1:52-54. The '855 Patent discloses OO modeling stored in database tables that store (1) object types called classes, (2) class attributes, and (3) relationships between classes. Ex. 1003, ¶¶25-27. Figure 2 depicts a "class table 150," which stores classes, and "[a] class_attribute table 112 [that] may associate a given attribute to one or more classes (in class table 150)." Ex. 1001, 5:10-11. Multiple classes can have the same attribute (a well-known feature of class inheritance) using the "class_attribute table." *Id.*, 5:11-14; Ex. 1003, ¶27.

B. Prosecution History of the '855 Patent

The '855 Patent was filed on June 3, 2011. Ex. 1001; Ex. 1003, ¶28. The independent claims were amended to overcome prior art in a March 16, 2012 examiner amendment, and imported the "element history module" and "element document module" limitations from the dependent claims. Ex. 1002, 32-33.

IV. LEVEL OF ORDINARY SKILL IN THE ART

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.[1] Ex. 1003, ¶23. Petitioner further proposes that more education could compensate for less experience and vice versa. *Id.*

V. CLAIM CONSTRUCTION

With the exception of "module", Petitioner has applied the ordinary meaning understood by a POSITA at the time of the alleged invention, in the context of the '855 Patent and its prosecution history. *Phillips v. AWH Corp.*, 415 F.3d 1303 (Fed. Cir. 2005).

A. "module"

The challenged claims terms recite several "modules." The '855 Patent specification describes the "data repository including" each of these "modules." *Id.*, 1:66-2:15. A POSITA would thus understand "module" to be a portion of a data repository. Ex. 1003, ¶30. Indeed, in the parallel litigation, counsel for Patent Owner conceded that "a table or a portion of a table" (*i.e.*, a data repository) is a "module." Ex. 1013, 78. Petitioner applies Patent Owner's own definition of "module" throughout this Petition.

[1] Dr. White surpasses this level of skill. *Id.*, ¶24.

VI. OVERVIEW OF THE PRIOR ART

A. Althoff (Ex. 1006)

Althoff is directed towards "object-oriented database applications, while retaining the ability to use present relational database management systems." Ex. 1006, 3:6-9; Ex. 1003, ¶¶44-45. Althoff discloses 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 "inheritance relationships and data-model relationships." *Id.*, 3:61-63.

B. Surjanto (Ex. 1008)

Surjanto discloses an "object-relational database management system" that "improves content management of XML documents" by "exploiting their structural information." Ex. 1008, Abstract, Ex. 1003, ¶¶46-47. In particular, Surjanto discloses that "[w]ith the advent of object-relational database management systems (ORDBMSs) [Sto98], developers can extend the database system." *Id.*, 9. Surjanto "develop[ed] XCoP as part of an integrated Web database framework ... built … using the extensibility features of an ORDBMS." *Id.*, 9-10. Surjanto provides benefits such as content reuse, exploitation of structural information, content consistency enforcement, etc. *Id.*, 10; Ex. 1003, ¶¶46-47.

Surjanto published in 2000, and is a printed publication that qualifies as prior art. As shown on the cover page of Exhibit 1008 and as set forth in the accompanying Dr. Hall-Ellis declaration, Surjanto qualifies as a printed publication because it was published in the Proceedings of the First International Conference on Web Information Systems Engineering ("WISE"). Ex. 1008, 1-2; Ex. 1010, ¶39. Surjanto was made available in university libraries including the Linda Hall Library of Science, Engineering & Technology. Ex. 1010, ¶41. A copy from that library's online catalog and records thereof demonstrate that the conference proceedings were available as of October 6, 2000. *Id.*, ¶¶41-42. Library records from Columbia University Libraries also indicate that a copy of the conference proceedings were available at that library on October 6, 2000. *Id.*, ¶43. Further supporting the public availability of these proceedings is that a copy of these conference proceedings was indexed by the Library of Congress. *Id.*, ¶¶44-45. Based on this and other library records, Dr. Hall-Ellis concludes that Surjanto "was published and publicly accessible by October 6, 2000." Ex. 1010, ¶45.

The well-known nature of IEEE publications is confirmed by Dr. Jules White. Dr. White states in his declaration that he is familiar with the WISE conference and its publication practices, and that (as with all IEEE-published papers) papers presented at the WISE conference would have been published in its proceedings and readily available as of the publication of those proceedings. Ex.

1003, ¶¶49-52. Moreover, Surjanto and the proceedings it appears in bears a 2000 date throughout the publication, further indicating its reliability. Ex. 1008, 1-2, 9. Dr. White also confirms that Surjanto is available on IEEE Xplore (https://ieeexplore.ieee.org/abstract/document/882377), which states that a copy has been available on that site since August 6, 2002. Ex. 1003, ¶50. Dr. White testifies that the IEEE Xplore provides reliable dates indicative of their publication status. *Id.*

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. 1008, 1-2; Ex. 1003, ¶¶49-52. Indeed, the Board has stated that the "IEEE is a well-known, reputable compiler and publisher of scientific and technical publications, and we take Official Notice that members in the scientific and technical communities who both publish and engage in research rely on information published on the copyright line of IEEE publications." *Ericsson Inc. v. Intellectual Ventures I LLC*, IPR2014-00527, Paper 41, at 11 (PTAB May 18, 2015). The Board has further found that "[a]llowing IPR petitioners to rely on the IEEE publication date in an IPR proceeding … serves the interests of justice." *Id.*, 12. Dr. White agrees with these statements from the Board. Ex. 1003, ¶51.

Moreover, Exhibit 1008 is an authentic copy of Surjanto as it was published in 2000. Ex. 1003, ¶52; Ex. 1010, ¶39. Both Dr. White and Dr. Hall-Ellis have examined Exhibit 1008 and, based on their experience with the reliability of published dates within IEEE-publications and the lack of any visible alterations, conclude it is the authentic copy as published in 2000 in the WISE proceedings. Dr. White also confirms that the Surjanto paper within Exhibit 1008 is identical to the version available on IEEE Xplore, which was publicly available in 2002.

Further, pursuant to FRE 901(b)(8), Exhibit 1008 is authentic because it is an "ancient document."

C. Nye (Ex. 1005)

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. *Id.*, 12:39-41; Ex. 1003, ¶¶53-55.

Figure 4 depicts the system for the event model, events, and event states.

D. Moore (Ex. 1007)

Moore teaches a system that "take[s] a property that is stored in the [relational] 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. *Id.*, 5:51-54; Ex. 1003, ¶¶56-57.

VII. DETAILED EXPLANATION OF INVALIDITY GROUNDS

A. Ground 1: Althoff in view of Surjanto renders obvious claims 1, 3-12, 14-23, 25-33

Ground 1 relies on the OO relational databases of Althoff modified to include the document management extensions taught by Surjanto to render obvious claims 1, 3-12, 14-23, and 25-33. Ex. 1003, ¶¶58-158.

1. Motivation to Combine

A POSITA would have found it obvious to modify Althoff's object-oriented relational database to include Surjanto's teachings of XML document management extensions. Ex. 1003, ¶¶59-69.

Specifically, a POSITA would have been motivated to combine Surjanto's teachings of XML document management extensions into Althoff's database to achieve 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; Ex. 1003, ¶61. By supporting Surjanto's extensions, developers could, for example, build XML-supporting web applications using Althoff's database. Ex. 1003, ¶67.

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 other words, Surjanto teaches how to extend OO databases to support storage and manipulation of complex XML documents, both their text and their structure. Ex. 1003, ¶66.

A POSITA would be motivated to modify Althoff to include Surjanto's XCoP extensions because of the well-known benefits of XML document support. Ex. 1003, ¶¶158-64. Surjanto itself notes XCoP allows "users to reuse and process textual portions of document contents." Ex. 1008, 9. 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.); Ex. 1003, ¶¶66-67.

One reason a POSITA would have incorporated Surjanto is because it enables developers to build web applications on top of the Althoff database. Ex. 1003, ¶67. A web application is application software that is accessed over the internet using a web browser. *Id.* In the 1990s and early 2000s, the use of web applications was rapidly growing. *Id.* A POSITA would have been motivated to integrate Surjanto's XML extensions into the database system of Althoff to extend the types of applications that could be built on top of the database to include web applications. *Id.* 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; Ex. 1003, ¶67. 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." *Id.* 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. Ex. 1003, ¶¶65-67.

Also, Althoff is exactly the type of object-relational database similar to ORDBMS databases which Surjanto sought to extend. Ex. 1003, ¶65. Surjanto notes 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*, 9-10. Althoff likewise discloses "object-oriented database structures," including "[a] meta-model of the user's object database," provides "a set of classes, objects, and relationships between classes." Ex. 1006, Abstract. A POSITA would readily understand Althoff's "object-oriented database" is a type of the "object-relational database management systems" that Surjanto sought to extend. Ex. 1003, ¶61-65, Ex. 1008, 9.

A POSITA would have combined Althoff and Surjanto with a reasonable expectation of success. Ex. 1003, ¶68. Althoff's OO database and database model was compatible with Surjanto, as expressly taught by Surjanto. *Id.* 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. Ex. 1003, ¶68.

Moreover, the combination of Althoff and Surjanto would have yielded predictable results. Ex. 1003, ¶69. 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. Ex. 1003, ¶¶66-67. Indeed, Surjanto explains that this result would be predictable because "XcoP is flexibly configurable." Ex. 1008, Abstract. Thus, Surjanto would have been readily combined with other database platforms like Althoff to achieve the predictable benefits taught by Surjanto. Ex. 1003, ¶66-67.

Additionally, Althoff and Surjanto are both directed to the same endeavor. Ex. 1003, ¶60. Althoff and Surjanto each disclose a data model for a database system. *Id.* Althoff specifically teaches representing data and relationships from the OO model to a relational database. *Id.* Surjanto specifically teaches a database management system that improves content management of documents. *Id.* 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, 1:52-54. All three references, therefore, are directed to the same field of endeavor, and are analogous art. Ex. 1003, ¶60.

2. Claim 1

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

A computerized system for manipulating data sets: To the extent the preamble is limiting, Althoff discloses this limiation. Ex. 1003, ¶¶70-71. 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. A POSITA would understand that Althoff's object database includes ***datasets***. Ex. 1003, ¶71.

(b) 1[a]: a hardware-based processor; and

[A] hardware-based processor: Althoff discloses this limitation. Ex. 1003, ¶¶72-73. Althoff teaches "using a general purpose ***computer***, or a set of general purpose computers." Ex. 1006, 5:21-31. A POSITA would understand that a general purpose computer includes a central processing unit, also known as a ***hardware-based processor***. Ex. 1003, ¶73.

(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:

[A] data repository: Althoff discloses the data repository as a "relational ***database*** ... implemented using a relational database engine." Ex. 1006, 6:42-55. A POSITA would understand that a relational database is a specific implementation of a ***data repository***. Ex. 1003, ¶¶74-80.

[T]hat contains the data sets and one or more layers of metadata of the data in the data sets: Althoff teaches a "user's object ***database***." Ex. 1006, 3:25-29. A POSITA would understand that an object database ***contains data sets*** in the form of objects and the data associated with those objects. Ex. 1003, ¶¶74-75, 77. 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. 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***). Ex. 1003, ¶¶79-80.

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

[A] class module: As discussed below, the ***elements*** stored in the ***element module*** are instanced objects of a class. *See* Ex. 1003, ¶¶81-84; 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 elements*** (for example, a Manufacturer class of the plurality of Manufacturer objects). Ex. 1003, ¶82.

[C]onfigured to define at least one class of elements: Althoff teaches that these class modules are defined and stored in a "class Class 301" table (*i.e.*, ***class module***). Ex. 1003, ¶83. 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.

[C]onfigured to store the class: 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; Ex. 1003, ¶¶83-83. Such information includes the "name to be displayed for the class" and a "description of the class." Ex. 1006, 11:18-48.

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

[A]n attribute module configured to define one or more attributes: 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; Ex. 1003, ¶85. A POSITA would understand that in OO software development models, ***attributes are collections of data*** associated with each class. Ex. 1003, ¶86. 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. Ex. 1003, ¶87. Indeed, Figure 3 of Althoff teaches that a "Class Property" object "is an attribute of" a "Class" object. *Id.*

[A]n attribute module configured to store the attributes: 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. Ex. 1003, ¶¶88-89. 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; Ex. 1003, ¶89.

(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;

[C]lass-attribute associations between at least one of the attributes and the class: Althoff discloses relationships between classes, referred to as class objects, and attributes, referred to as searchable properties of class objects. Ex. 1003, ¶¶90-95. 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. A POSITA would understand that relationships (*i.e.*, "relationship 103") in object oriented-programming databases encompass both associations, which describes how object use or contain other objects, and inheritance relationships, which describes "is a" relationships between objects (*e.g.*, an F150 "is a" Ford Truck). Ex. 1003, ¶91. 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.[2] Ex. 1003, ¶92

[2] 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.

[A] class attribute module configured to define and store one or more class-attribute associations: This limitation is met by Althoff's "relationships 103." Ex. 1003, ¶93. 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. A POSITA would understand that the data model relationships between classes and attributes ***define and store the class-attribute associations***. Ex. 1003, ¶94.

Althoff also discloses a "class Pointer Valid Class 305," which is also a ***class attribute module***. Ex. 1003, ¶108. 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.*** Ex. 1003, ¶95.

(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;

[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. Ex. 1003, ¶¶96-100. 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; Ex. 1003, ¶97. As discussed above for limitation 1[e], searchable properties are attributes for a class of objects. 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***).

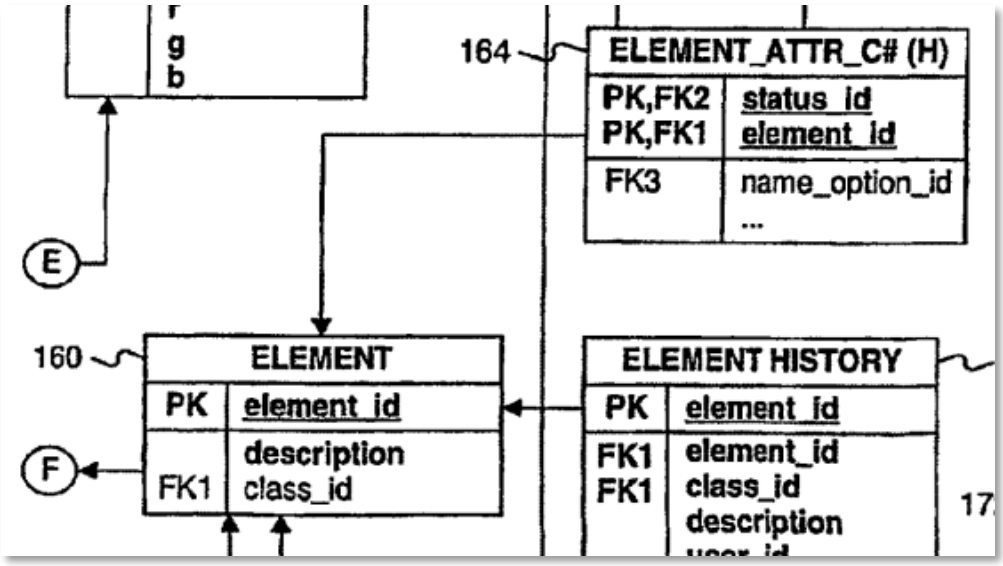
[T]ransmit the attributes and elements: Althoff discloses that the user database model 230 transmits attributes and elements. Althoff teaches that "the system 200 translates the user model 230 into a user relational database 231" via a process described by Figure 7. Ex. 1006, 8:18-20. This is done "using 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. Ex. 1003, ¶98.

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***). Ex. 1003, ¶99.

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. Ex. 1003, ¶100. 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. *Id.* This disclosure is similar to what is described in Althoff (indeed, Althoff expressly discloses that the table is populated using transmitted SQL commands). *Id.*



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;

[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. Ex. 1003, 101-106.

[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; Ex. 1003, ¶103. Thus, a Althoff's individual class tables are ***element modules*** because each row of the table stores objects (*i.e.*, ***elements***). Ex. 1003, ¶¶102-103.

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. Ex. 1003, ¶104. 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.

Figure 8 depicts an example of these individual class tables. Ex. 1003, ¶105. 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.

[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***). Ex. 1003, 106.

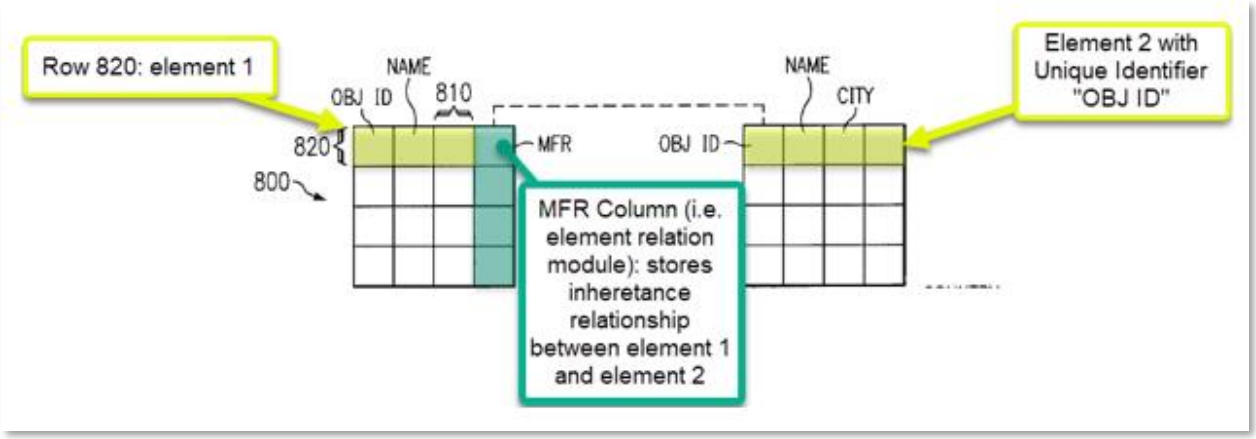
(i) 1[h]: an element relation module configured to store one or more relationships between the elements in the element module;

[A]n element relation module: Althoff discloses ***an element relation module*** as columns in a class table that store the relationship between each class and a related class. Ex. 1003, ¶¶107-112. As discussed above, Althoff's system creates a table for each class which is populated with instance objects of that class corresponding to each row of the table. Althoff discloses that Figure 7 is an example of those tables. Ex. 1003, ¶108; 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.").

[C]onfigured to store one or more relationships: 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 id.*, 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."). Thus, column 810 (*i.e.*, the ***element relationship module***) stores inheritance relationships between objects. Ex. 1003, ¶¶107-110.

[B]etween the elements in the element module: 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. Ex. 1006, 24:9-31; *see also id.*, 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."); Ex. 1003, ¶¶107-110. 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." Ex. 1006, 30:37-41.



Id., Fig. 8 (annotated).

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.*; Ex. 1003, ¶110. 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." Ex. 1006, 24:26-31; Ex. 1003, ¶110.

Althoff further teaches a "class Class Link 302," which is also an ***element relation module***. Ex. 1003, ¶111. 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." Ex. 1006, 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. 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; Ex. 1003, ¶112. 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. Ex. 1003, ¶112. 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." *Id.*; Ex. 1006, 6:12-16 ("[T]he class 101 component has a data-model relationship with the class 101 manufacturer ... each object of the class 101 component has a related object of the class 101 manufacturer."). Thus, the "class Class Link 302" ***stores one or more relationships among the elements stored in*** tables—including ***the element module*** which is defined in Althoff as a specific implementation of a table (*i.e.*, "results table 262"). Ex. 1003, ¶112.

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

[A]n element history module configured to store one or more indicators of a modification to the elements: In the Althoff-Surjanto combination, this limitation is disclosed by Surjanto's fragments. Ex. 1003, ¶¶113-116. 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.*, 11. 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***). Ex. 1003, ¶114. 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. Ex. 1003, ¶115.

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. *Id.*, ¶116. 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.

[A]n element document module configured to store formatting data for grouping a subset of the elements: In the Althoff-Surjanto combination, this limitation is disclosed by Althoff's class Class Configuration. Ex. 1003, ¶¶117-118. 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 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.

3. 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

[A] type definition module configured to define and store one or more types of the class: Althoff's Class Type, Property Type, and Object Type columns are the ***type definition module***. Ex. 1003, ¶¶119-1128. As described above, Althoff's class Class table defines and stores classes of the elements. *Supra*, §VIII.A.2(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; Ex. 1003, ¶119.

[A] type definition module configured to define and store the attributes related to the class: As described above, Althoff's class Class Property defines and stores the attributes related to the class. *Supra*, §VIII.A.2(e). 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; Ex. 1003, ¶120.

[A] type definition module configured to define and store the relationships between the elements: 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." Ex. 1006, 23:13-16; Ex. 1003, ¶121. 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. 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. Ex. 1003, ¶122. 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). *Id.*

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. Ex. 1003, ¶¶113-118. 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; Ex. 1003, ¶¶123-124. 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 type of relationship between the "pointing and target" as type "is-manufactured-by". 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 relationships between the elements, as claimed. *Id*.

Althoff's class Pointer Valid Class discloses ***a type definition module configured to define and store the attributes related to the class***, as claimed. Ex. 1003, ¶115. As described above, Althoff's Pointer Class Property ***define and store attributes***. *Supra*, §VIII.A.2(e). 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; Ex. 1003, ¶125.

Finally, Althoff's class Pointer Valid Class ***defines and stores the relationships between the elements,*** as claimed. Ex. 1003, ¶¶126-128. As noted above, Althoff discloses elements as objects (*i.e.* rows) in the class tables. *Supra*, §VIII.A.2(c). A POSITA would understand that elements (*i.e.*, objects) have properties of their class, such as type. Ex. 1003, ¶127. Storing one or more types of

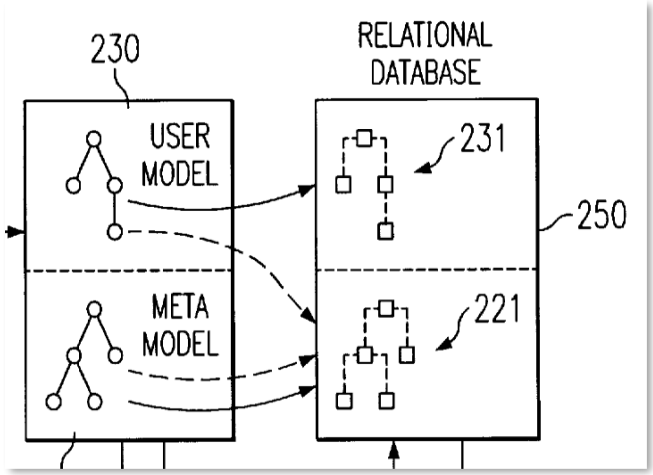
the class and the relationships between classes also ***defines and stores the relationships between the elements*** that belong to those classes. *Id.*

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

[A] list option module configured to enumerate one or more picklist attributes of a picklist type: In the Althoff-Surjanto combination, this limitation is disclosed by Althoff's Enumerated Valid Value and Range Valid Value. Ex. 1003, ¶¶129-130. 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.*

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; Ex. 1003, ¶131. Further, as shown below in Figure 2, Althoff discloses that the ***meta-model is stored in the relational database***.



Id., Fig. 2.

4. 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.

[A] named text module configured to store textual representations of query, control, and display mechanisms that relate to the elements: In the Althoff-Surjanto combination, this limitation is rendered disclosed by Althoff's query and access control model. Ex. 1003, ¶¶132-36. 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.*

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. 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. Ex. 1003, ¶134.

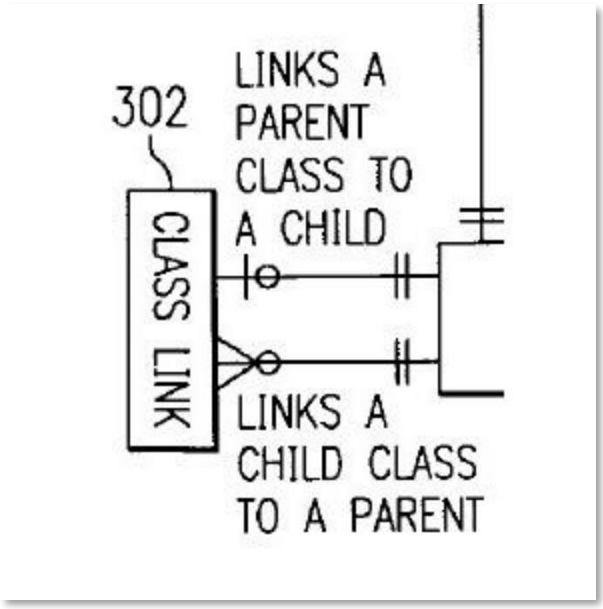
Moreover, Althoff discloses that "***predefined classes***" are used "for ***modeling access control*** to aspects of the user's object database" so that "[a]ccess may be restricted to selected users or selected user groups, and may be specified for individual objects, classes of objects or properties 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. Ex. 1003, ¶135. 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. *Id.*

Further, 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." *Id.*, ¶136.

5. 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.

[A] tuple module configured to generate and store one or more tuples: In the Althoff-Surjanto combination, this limitation is disclosed by Althoff's Class Link. Ex. 1003, ¶¶137-140. 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. 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. Ex. 1003, ¶138.



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. *Id*. 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

Ex. 1006, Tab. 3-3.

[W]herein the tuples associate subsets of the elements in accordance with types associated with the subsets of the elements: Althoff's objects in the Class Link disclose this limiation. Ex. 1003, ¶139. As explained above, Althoff teaches that elements are objects of a given Class class. 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. 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. Ex. 1003, ¶140. 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***. *Id.*, ¶140.

6. 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.

[W]herein the tuple module link[s] a first folder to at least one additional folder in a hierarchy: In the Althoff-Surjanto combination, this limitation is disclosed by Surjanto's table hierarchy. Ex. 1003, ¶¶141-144. 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, 16. A POSITA would understand that Surjanto's table hierarchy ***links folders that contain elements based on the type hierarchy*** defined for each sub-class of elements. Ex. 1003, ¶142.

[W]herein the tuple module uses recursion to link: Recursion is a fundamental concept in computer programming that has existed for decades, and a POSITA 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***. *Id.*, ¶143. Indeed, Surjanto teaches the use of recursion in a similar context for creating object child fragments (*i.e.*, fragments that are part of a parent-child hierarchy) 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, 13. Thus, a POSITA would understand that the fundamental programming concept of recursion—which is used

in Surjanto for creating child fragments in a hierarchy—would be particularly well-suited for linking folders that contain sub-classes of elements in Surjanto's table hierarchy—and as described below, would have also been applicable to Althoff's tuple module which functions in a nearly identical way. Ex. 1003, ¶143.

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. *Id.*, ¶144. 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***. *Id.*; Ex. 1006, 12:29-34.

7. 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.

[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: In the Althoff-Surjanto combination, this limitation is disclosed by Surjanto. Ex. 1003, ¶¶145-146. Surjanto teaches a "supertype" for each subtype that corresponds to a sub-class. Ex. 1008, 16. 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***). Ex. 1003, ¶146. A POSITA would further understand that Surjanto's ***data repository*** teaches ***filtering the one or***

more entries into one or more folders because the supertype encompasses a hierarchy of folders for various sub-classes of elements that are organized by subtypes in Surjanto's table hierarchy. *Id.*, ¶146.

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. *Id.*, ¶146. 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.*, ¶146.

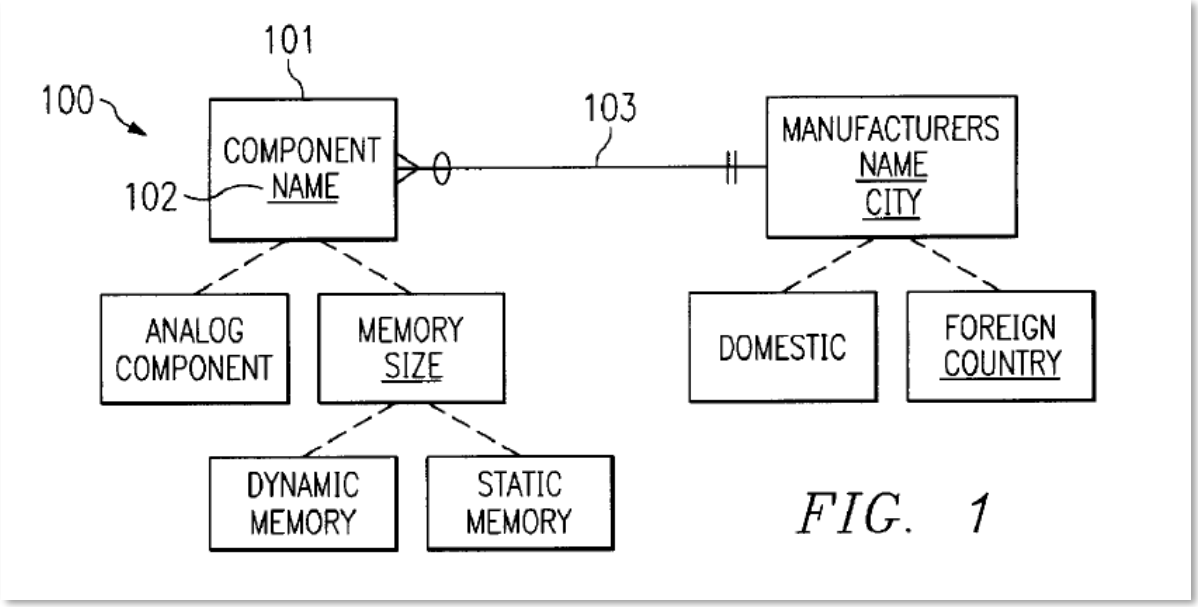
8. Claim 8: The computerized system of claim 1, wherein the class attribute associations are identified by logical names and class names.

[W]herein the class attribute associations are identified by logical names and class names: In the Althoff-Surjanto combination, this limitation is disclosed by Althoff. Ex. 1003, ¶¶147-148. Althoff's Class Property objects and Pointer Class Property objects are the attributes, and Althoff's class Class objects are the classes. Ex. 1003, ¶147. 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; Ex. 1003, ¶147. 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." Ex. 1006, Tabs. 3-2, 3-4; Ex. 1003, ¶147. 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. Ex. 1003, ¶148.

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

[W]herein the element attribute module is identified using a name of the class: In the Althoff-Surjanto combination, this limitation is disclosed by Althoff. Ex. 1003, ¶¶149-150. As explained above, Althoff's relational class tables are the ***element attribute module***. 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. 1003, ¶149.



Ex. 1006, Fig. 1.

Each class is represented by a relational database class table of the same ***name of the classes***: "table 800 component, a table 800 analog component, a table 800 memory, a table 800 dynamic memory, a table 800 static memory, a table 800 manufacturer, a table 800 domestic manufacturer, and a table 800 foreign manufacturer." *Id.*, 23:67-24:8. Indeed, for each "table 800," the system creates "a column 810 'name'." *Id.*, 24:18-19. Thus, Althoff's relational class tables (*i.e.*, ***the element attribute module***) are each ***identified using the name of the class that they represent***. Ex. 1003, ¶150.

10. 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.

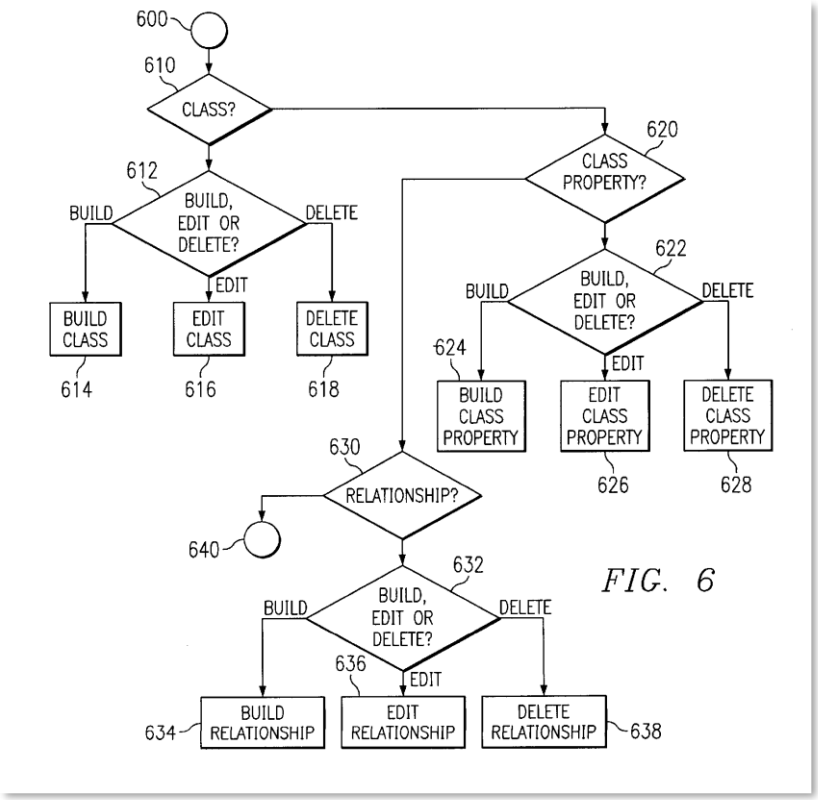
[W]herein the processor is configured to generate a data model of the elements based on the layers of metadata: In the Althoff-Surjanto combination, this limitation is disclosed by Althoff's meta-model. Ex. 1003, ¶¶151-52. As described above, Althoff discloses a general purpose computer that includes a hardware-based ***processor*** and ***data sets*** that are ***stored*** in the "user's object database" (*i.e.*, ***the data repository***). 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. Thus, 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 metadata***, when the object database is translated into a set of relational database structures. Ex. 1003, ¶152.

11. 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.

[W]herein 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: In the Althoff-Surjanto combination, this limitation is disclosed by Althoff's GUI. Ex. 1003, ¶¶153-55. 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***). 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; Ex. 1003, ¶153.

Althoff further teaches that "[t]he functions provided by the GUI 211 ... are further described with regard to FIG. 6." Ex. 1006, 7:19-22. 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. Ex. 1003, ¶154. A POSITA would understand that Althoff's GUI is ***based on the data in the data sets and the layers of metadata stored in the data repository*** because the interface displays information about the object-database that is included in Althoff's meta-model. *Id.*, ¶155.



Ex. 1006, Fig. 6.

12. Claims 12, 14-23, 25-33

Claims 12, 14-23, and 25-33 mirror claims 1 and 3-11, as set forth in the following chart. The combination of Althoff and Surjanto renders these claims obvious for the same reasons as set forth above for claims 1 and 3-11. Ex. 1003, ¶156.

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

Claim Limitation	**Corresponding Claim Limitation(s)**
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[pre]	3[pre]
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]
23[b]	1[d]
23[c], 23[d]	1[e]

Claim Limitation	Corresponding Claim Limitation(s)
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[pre]	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

B. Ground 2: Althoff in view of Surjanto in further view of Nye renders obvious claims 2, 13, 24

Ground 2 further modifies the Althoff-Surjanto combination with Nye's event model. The resulting combination renders obvious claims 2, 13, and 24. Ex. 1003, ¶¶157-177.

1. Motivation to Combine

A POSITA would have found it obvious to further modify the combination of Althoff's database and Surjanto's document management extensions with Nye's teachings of an event model. Ex. 1003, ¶¶158-170. Specifically, a POSITA would have been motivated to combine Nye's teachings of an event model into Althoff's database to achieve the known benefits of combining a state machine with the OO model. *Id.*, ¶¶161-166. 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. Since Nye's event model models state transitions, a POSITA would understand that it is a state machine. Ex. 1003, ¶161. A POSITA would understand that the integration of state machines into the OO model taught by Althoff provides many additional benefits. One such benefit is that state machines are a very clear way to describe behavior of a system. At a high-level, a state machine allows a developer to describe the different "modes" of operation of the system and how the system transitions through each mode. Ex. 1003, ¶¶161-165. A POSITA would have also understood that a state machine has the added benefit of making an OO database more robust, easier to maintain and work with, as a state machine presents data in a far more readable

fashion than underlying software source code. *Id.* Likewise, state machines describe the behavior of the system independently of the code. *Id.* 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. *Id.* Given the known benefits of state machines, it is unsurprising that they were commonly combined with the OO model. *Id.*

As explained by Dr. White, it had been "long-standing practice to integrate state machines such as the one taught by Nye's event model into object-oriented software development systems." *Id.* Such state machines were long-known and common in the field of software development. *Id.* 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. *Id.*

A POSITA would have been further motivated to combine Nye's teachings of an event model into Althoff's relational database improved by Surjanto's document management extensions because it would further Althoff's goal of representing the OO model in a relational model. Ex. 1003, ¶165; Ex. 1006, 3:4-9 ("[I]t 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 system."). A POSITA knew that state machines provided such advantages. Ex. 1003, ¶¶165-166

A POSITA would have combined Althoff, Surjanto, and Nye with a reasonable expectation of success. Ex. 1003, ¶¶167, 69. The use of Nye's event model was compatible with Althoff's objected-oriented database combined with Surjanto's document management extensions, as Nye's event model was used with "relational databases, object-oriented databases, or other database environments." *Id.*; Ex. 1005, 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. Ex. 1003, ¶169.

Moreover, the combination of Althoff, Surjanto, and Nye would have yielded predictable results. A POSITA would have readily understood that combining Althoff and Nye would enable event modeling in Althoff's database, and would thus further the functionality of the combination of Althoff's and Surjanto's systems. *Id.*, ¶170. Further, a POSITA would have understood that this result would be predictable because Nye specifically discloses how to implement the event model in other databases and the results that would be achieved in existing database environments, such as the environment taught by the combination of Althoff and Surjanto. *Id.*

Moreover, Althoff, Surjanto, Nye, and the '855 Patent are analogous art because they are all directed to the same field of computer data storage, more specifically storing and managing data sets in a database system. Ex. 1001, 1:61-65; Ex. 1003, ¶160.

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

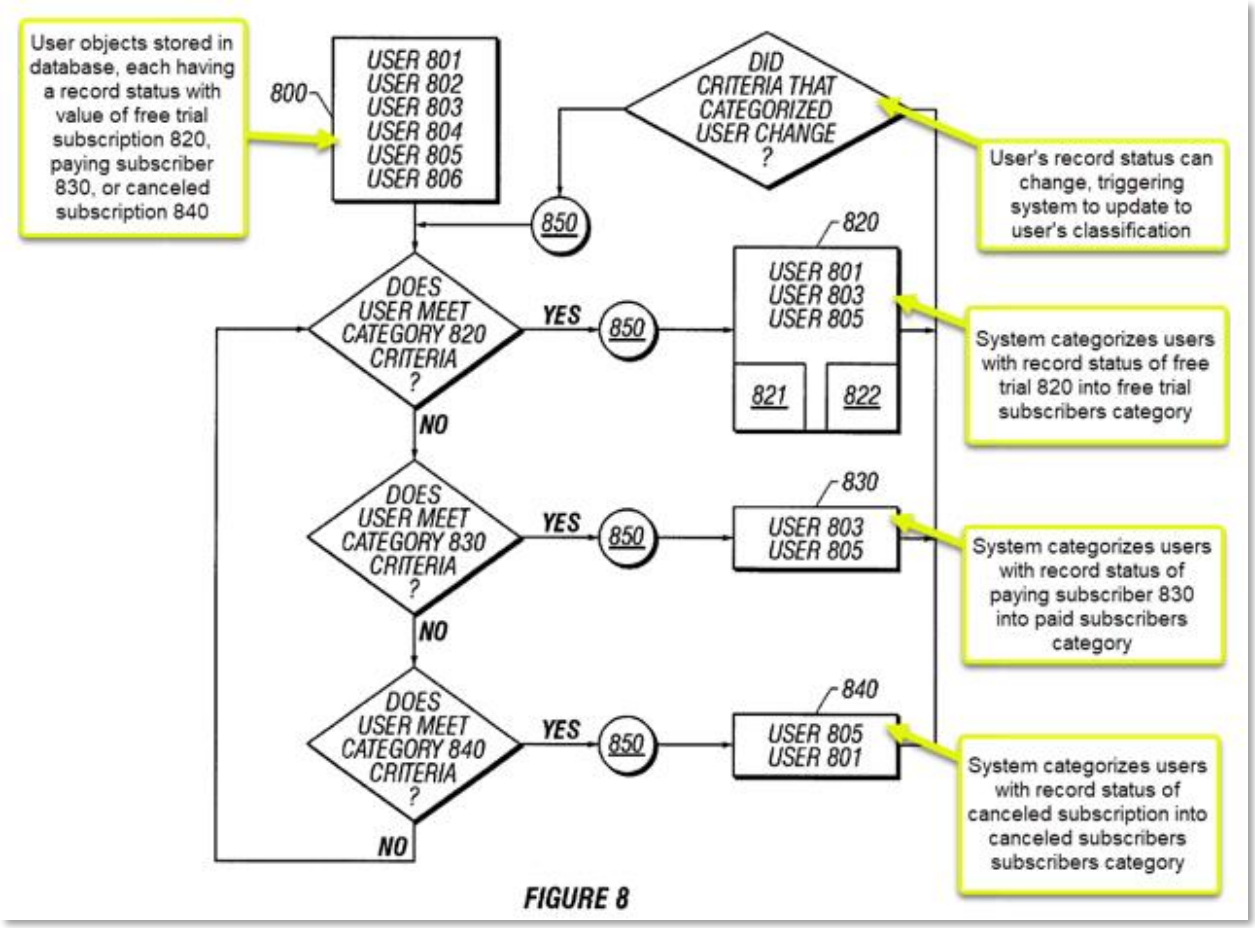
(a) 2[a]: a state machine module including a state machine transition module and a status module configured to define and store a state machine type,

[A] state machine module including a state machine transition module configured to define and store a state machine type: In the Althoff-Surjanto-Nye combination, this limitation is disclosed by Nye's event model. Ex. 1003, ¶¶171-17519-120. Nye teaches that the elements, referred to as records in a database, may be updated and that the "***status of records in a database change over time***." Ex. 1005, 5:28-42. Nye refers to a status of a record as the "state" of a record. *Id.* Nye further teaches that the event model "***provides a way to flexibly handle state transitions***." *Id.* For example, Nye teaches that "if a set of events relating to credit card authorization system is modeled, then the developer might choose to create the following event states: 'authorization approved,' 'authorization declined,' 'hardware error—cannot retry,' 'line is busy—retry later.'" *Id.*, 12:39-46. Thus, Nye's event model is a ***state machine module*** that represents the various event states and the transitions between those states. Ex. 1003, ¶¶171-175. And Nye's state transitions

describe the functionality of ***a state machine transition module*** included in the state machine module. Nye also teaches ***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.

[A] state machine module including a status module configured to define and store a state machine type: In the Althoff-Surjanto-Nye combination, this limitation is disclosed by Nye's record state. Ex. 1003, ¶¶171-175. 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.*, ¶174; 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.

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. 1003, ¶175.

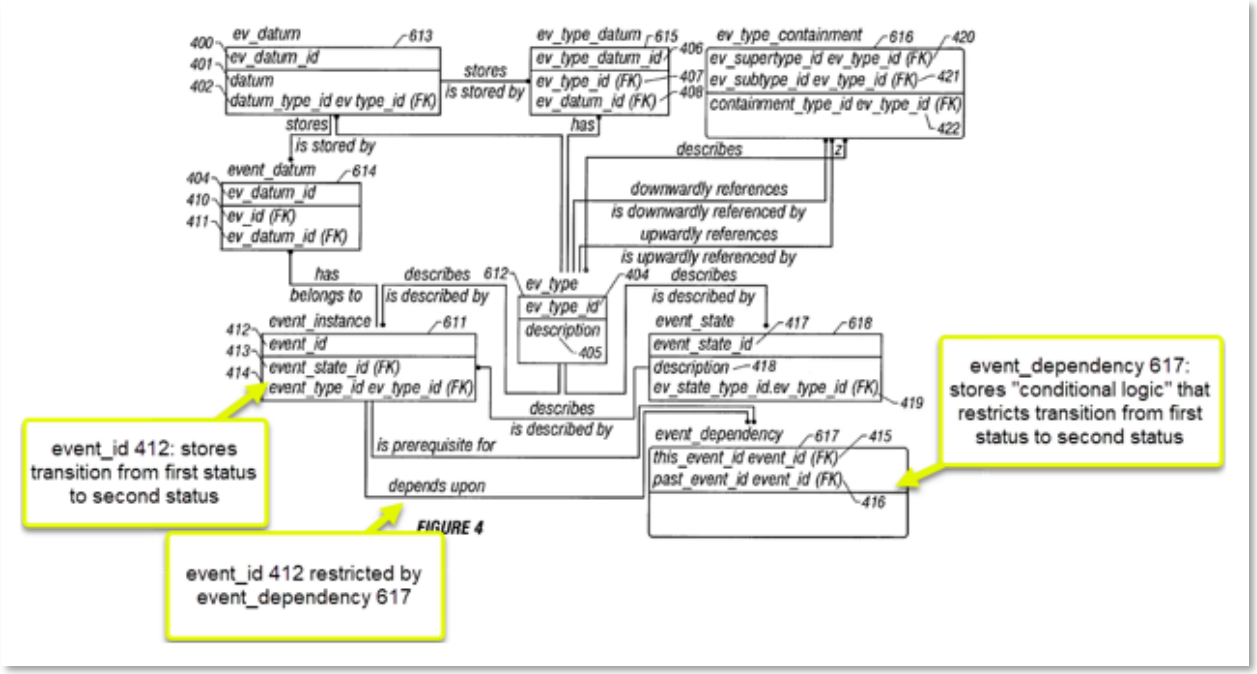


Ex. 1005, Fig. 8 (annotated).

(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.

[T]he state machine type storing a first status of one element and a transition to a second status of the one element: In the Althoff, Surjanto, Nye combination, this limitation is disclosed by event_dependency table. Ex. 1003, ¶176. 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. *Id.* 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, 24

Claims 13 and 24 mirror claim 2, as set forth in the following chart. The combination of Althoff, Surjanto, and Nye renders these claims obvious for the same reasons as set forth above for claim 2. Ex. 1003, ¶177.

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]
24[b]	2[a]

C. Ground 3: Althoff in view of Surjanto in further view of Moore renders obvious claims 4, 6-7, 15, 17-18, 26, 28-29

The combination of Althoff, Surjanto, and Moore renders obvious claims 4, 6-7, 15, 17-18, 26, and 28-29. Ex. 1003, ¶¶179-188. While, per Ground 1, these claims are rendered obvious by Althoff and Surjanto, to the extent, Patent Owner argues Althoff and Surjanto do not disclose the "named text module" and "tuple module," these limitations are also disclosed by Moore.

1. Motivation to Combine

A POSITA would have found it obvious to combine Althoff and Surjanto with Moore's virtual folder system. Ex. 1003, ¶¶179-188. 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). *Id.*, ¶181. 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. *Id.*

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.

A POSITA would be motivated to incorporate Moore's teachings of a virtual folder system with Althoff and Surjanto 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 OO database is used to store files or other items like contacts, and where Surjanto's table hierarchy discusses organizing sub-classes and sub-types of elements. Ex. 1003, ¶¶181-84. Indeed, Moore notes that its virtual folder 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. 1006, 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.*, 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.*, 3:41-45.

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. Ex. 1003, ¶184. 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. *Id.* 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. *Id.* 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. Ex. 1003, ¶84.

A POSITA would have combined Althoff, Surjanto, and Moore with a reasonable expectation of success. Ex. 1003, ¶186. The use of Moore's virtual folder system was compatible with relational databases and database structures, including those disclosed in Althoff and Surjanto. *Id.* Moore provides numerous implementation details for incorporating a virtual folder system with a databases and database structures, including the use of various components in Moore's Figure 2. *Id.* 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. *Id.* A POSITA would understand that Moore was compatible with Althoff's relational database structure combined with Surjanto's document extensions, since Moore discloses that it queries a relational database, which is the type of database disclosed in Althoff as modified by Surjanto. *Id.*, ¶¶186-187.

Moreover, the combination of Althoff, Surjanto, and Moore would have yielded predictable results. Ex. 1003, ¶188. A POSITA would have readily understood that combining Althoff and Surjanto with Moore would result in database structures and XML document extensions as taught by Althoff and Nye with the capability of displaying the objects in the database using a virtual folder system as taught by Moore. *Id*. 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 the combination of Althoff and Surjanto, to achieve the predictable benefits taught by Moore.

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, 1:61-65. All four references, therefore, are directed to the same field of endeavor, and are analogous art. Ex. 1003, ¶180.

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.

[A] named text module configured to store textual representations of query, control, and display mechanisms that relate to the elements: In the Althoff-Surjanto-Moore combination, this limitation is disclosed by Moore's virtual folder system. Ex. 1003, ¶¶189-197. 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. *Id.*, ¶189. 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.

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

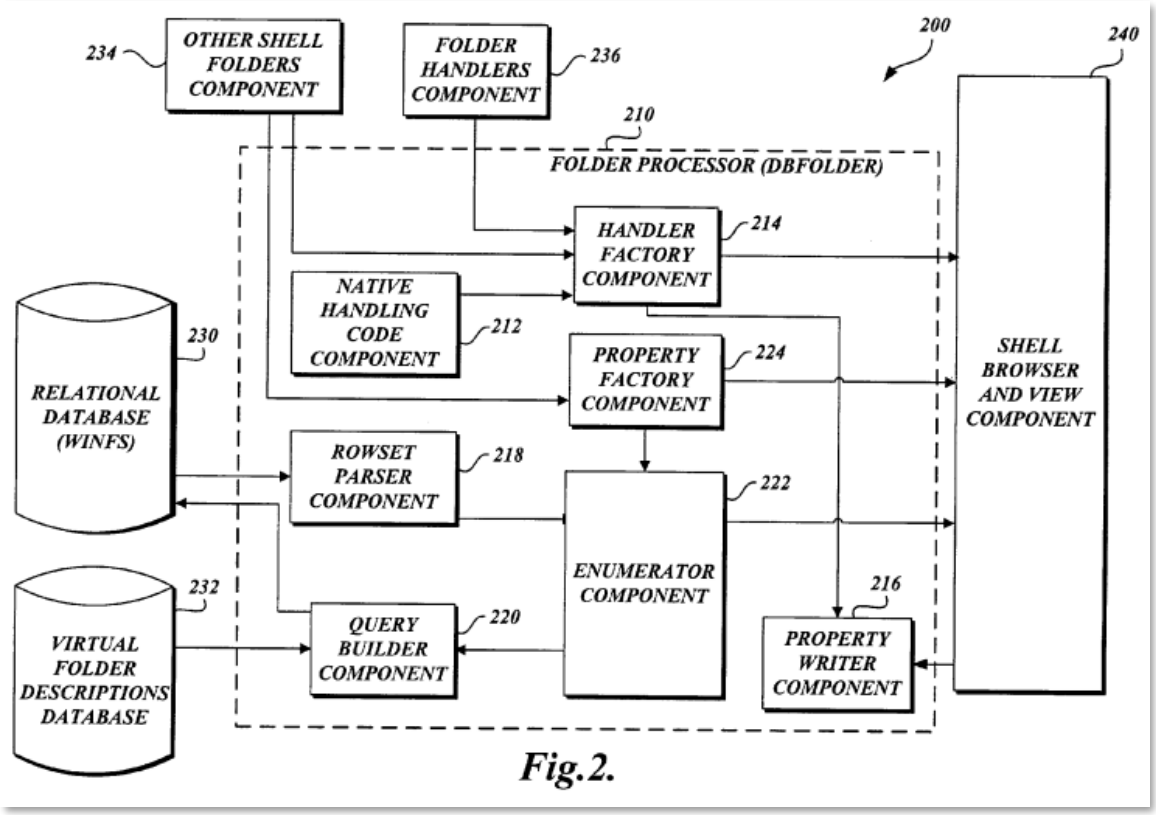


Fig.2.

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; Ex. 1003, ¶190.

The "virtual folder descriptions database" and "virtual folder system" together make up a ***named text module***. Ex. 1003, ¶¶191-192. 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." Ex. 1007, 8:24-28.

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.

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; Ex. 1003, ¶¶192-95. Thus, a SQL query is a ***textual representation of query, control, and display mechanisms that relate to the plurality of elements*** because a SQL query is a string (*i.e.*, ***textual representation***) query for filtering data (*i.e.*, ***plurality of elements***) in a database (*i.e.*, ***control mechanism***) that can also return a table of data for display (or feed the data to

another component that displays it) (*i.e.*, ***display mechanism***). Ex. 1003, ¶195. Because the query builder component builds SQL queries, it ***stores the textual representations*** (*i.e.*, SQL queries).

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; Ex. 1003, ¶196. 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. Ex. 1007, 8:24-28, 8:18-20. Thus, since the "virtual folder descriptions" is converted into a SQL query (*i.e.*, ***query, control and display mechanisms that relate to the plurality of elements***), the "virtual folder descriptions" is a ***textual representation*** of those mechanisms.

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

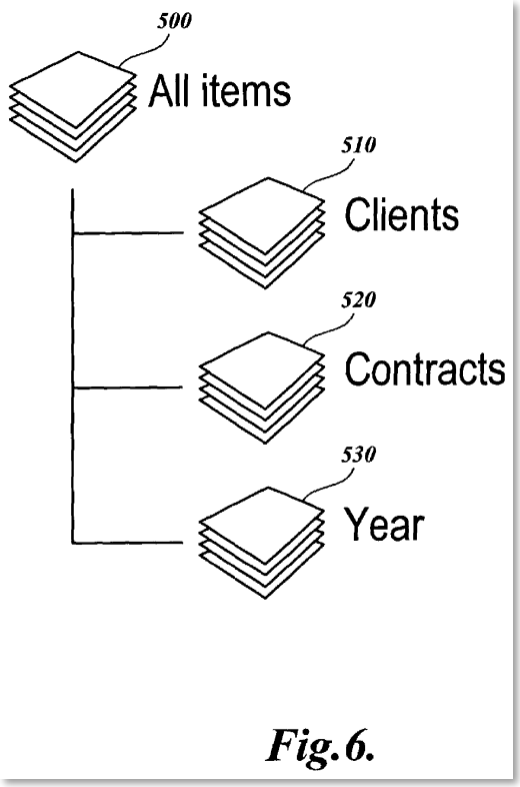
management system for document fragments (*i.e.*, ***elements***). Ex. 1003, ¶197. As already discussed, a POSITA would have known that Moore's virtual folder system of generating SQL queries for displaying of items in a virtual folder could be used with Althoff's database of objects and Surjanto's database management system. *Id.*

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.

[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, the virtual folder descriptions database ***links*** the folder descriptions (*i.e.*, ***a first folder to at least one additional folder***). Ex. 1003, ¶199.

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. 1003, ¶200.



Ex. 1007, Fig. 6.

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. Since the virtual folder can itself contain folders (and folders can further contain more folders), that results in ***folders linked in a hierarchy***. Ex. 1003, ¶201.

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; Ex. 1003, ¶202.

[W]herein the tuple module uses recursion to link: Recursion is a fundamental concept in computer programming that has existed for decades, and 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***. Ex. 1003, ¶203. 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. *Id*.

Indeed, as described above, Surjanto teaches the use of recursion in a similar context for creating object child fragments and their corresponding aggregation relationships. *Supra*, §VIII.A.6. 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). Ex. 1003, ¶204.

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.

[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, 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***). Ex. 1003, ¶¶205-207.

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). Ex. 1003, ¶194. 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; Ex. 1003, ¶¶206-207.

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

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*, VII.C; Ex. 1003, ¶208.

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

VIII. THE BOARD SHOULD NOT EXERCISE ITS DISCRETION TO DENY THE PETITION

The Board has discretion to deny institution. Petitioner hereby provides a *Sotera*-type stipulation and stipulates that if the PTAB institutes *inter partes* review in relation to any petition filed by ServiceNow with respect to the ’855 Patent, Petitioner will not pursue in the district court litigation any ground raised or that could have been reasonably raised in the instituted IPR with respect to the patent claim subject to the instituted IPR.

Regarding §325(d), the ’855 Patent has not previously been challenged at the PTAB. The examiner did not cite or consider any reference in this petition’s grounds. Discretionary denial under §325(d) and *General Plastic* is not warranted.

IX. MANDATORY NOTICES AND FEES

A. Real Party-In-Interest

The real party-in-interest for Petitioner is ServiceNow, Inc.

B. Related Matters

Petitioner has filed petitions against the related ’468 and ’585 patents, which are captioned IPR2023-01071 and IPR2023-01072.

Patent Owner is currently asserting the ’855 Patent against Petitioner in *InQuisient Inc. v. ServiceNow, Inc.*, Case No. 1:22-cv-00900-CJB (D. Del.).

C. Counsel and Service Information

Petitioner provides the following counsel and service information. Petitioner consents to electronic service to qe-servicenow-inquisient-iprs@quinnemanuel.com and the email addresses listed in the table below. Pursuant to 37 C.F.R. §42.10(b), a Power of Attorney accompanies this Petition.

LEAD COUNSEL	BACK-UP COUNSEL
James M. Glass (Reg. No. 46729) jimglass@quinnemanuel.com QUINN EMANUEL URQUHART & SULLIVAN, LLP 51 Madison Avenue, 22nd Floor New York, NY 10010 Tel: (212) 849-7000 Fax: (212) 849-7100	John T. McKee (Reg. No. 65,926) johnmckee@quinnemanuel.com QUINN EMANUEL URQUHART & SULLIVAN, LLP 51 Madison Avenue, 22nd Floor New York, NY 10010 Tel: (212) 849-7000 Fax: (212) 849-7100 Quincy Lu (Reg. No. 76,954) quincylu@quinnemanuel.com QUINN EMANUEL URQUHART & SULLIVAN, LLP 1109 First Avenue, Suite 210 Seattle, WA 98101 Tel: (206) 905-7000 Fax: (206) 905-7100 Joshua Parzivand (Reg. No. 80,650) joshuaparzivand@quinnemanuel.com QUINN EMANUEL URQUHART & SULLIVAN, LLP 555 Twin Dolphin Drive, 5th Floor Redwood City, CA 94065 Tel: (650) 801-5000 Fax: (650) 801-5100

D. Payment of Fees

The undersigned authorizes the Office to charge the fee required for this Petition for *Inter Partes* Review to Deposit Account No. 50-5708. Any additional fees that might be due are also authorized.

X. CONCLUSION

For at least the foregoing reasons, Petitioner requests institution of an IPR and cancellation of the Challenged Claims.

Date: August 28, 2023

Respectfully submitted,

By: /s/ *James M. Glass*
James M. Glass

Counsel for Petitioner ServiceNow, Inc.

CERTIFICATE OF COMPLIANCE WITH TYPE-VOLUME LIMITATION, TYPEFACE REQUIREMENTS, AND TYPE STYLE REQUIREMENTS

1. This Petition complies with the type-volume limitation of 14,000 words, comprising 13,873 words, as counted using the Microsoft Word software that was used to prepare this paper, excluding the parts exempted by 37 C.F.R. §42.24(a).

2. This Petition complies with the general format requirements of 37 C.F.R. §42.6(a) and has been prepared using Microsoft® Word 2016 in 14-point Times New Roman.

Date: August 28, 2023

By: */s/ James M. Glass*

James M. Glass
QUINN EMANUEL URQUHART
& SULLIVAN, LLP
51 Madison Avenue, 22nd Floor
New York, NY 10010
Tel: (212) 849-7000
Fax: (212) 849-7100

Counsel for Petitioner

CERTIFICATE OF SERVICE

Pursuant to 37 C.F.R. § 42.6(e), the undersigned hereby certifies that the foregoing document was served on August 28, 2023 by e-mailing copies to Patent Owner's counsel of record at:

IPR53604-0004IP1@fr.com

Date: August 28, 2023

By: */s/ James M. Glass*

James Glass
QUINN EMANUEL URQUHART
& SULLIVAN, LLP
51 Madison Avenue, 22nd Floor
New York, NY 10010
Tel: (212) 849-7000
Fax: (212) 849-7100

Counsel for Petitioner