

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

SERVICENOW, INCORPORATED,
Petitioners

v.

INQUISIENT INCORPORATED,
Patent Owner

Case IPR2023-01071
Patent 7,979,468

PATENT OWNER'S PRELIMINARY RESPONSE

TABLE OF CONTENTS

I.	INTRODUCTION	1
II.	SUMMARY OF THE '468 PATENT	5
III.	SUMMARY OF THE CITED REFERENCES.....	13
A.	Althoff (EX1006)	13
B.	Nye (EX1005)	17
IV.	CLAIM CONSTRUCTION	19
A.	Module.....	19
B.	The Petition Misstates Patent Owner's Construction of "Module"	20
V.	INSTITUTION SHOULD BE DENIED UNDER 37 C.F.R. § 42.104(B)(3) BECAUSE THE PETITION DOES NOT APPLY WHAT PETITIONER CONTENDS TO BE THE PROPER CONSTRUCTION OF THE TERM "MODULE"	21
VI.	GROUND 1 BASED ON ALTHOFF IN VIEW OF NYE IS DEFICIENT (CLAIMS 1-9, 11-22, 24-35, 37-39).....	25
A.	The Petition fails to identify a "class module" or its methods and, at the very least, fails to identify a "class module" in the same "data repository" as the "element module" (Limitations 1[e], 14[c], and 27[c])	25
1.	Summary	25
2.	The Petition maps the "data repository" to Althoff's relational database	27
3.	The Petition maps the "element module" of claim 1 to Althoff's tables in the relational database	27
4.	The Petition maps the "class module" to a class in Althoff's object database, rather than to a structure in Althoff's relational database	29
5.	There is no "Class 301 table" in Althoff.....	36

B.	The Petition fails to identify the “state machine module” and “status module” or their methods (Limitations 1[i], 1[j], 14[g], 14[h], 27[g], 27[h])	39
1.	The claimed invention requires two modules that incorporate state machines into the database	39
2.	Nye does not disclose a state machine.....	42
3.	The citations to Nye in the Petition offer no support for Petitioner’s assertions	46
C.	Dependent Claims	50
D.	Independent Claims 14 and 27	52
E.	No Motivation To Combine: Nye Teaches Away From The Proposed Combination	52
F.	No Motivation To Combine: The Petition Fails to Explain How Althoff’s Translation Process Would Work With Nye’s Event Model	54
VII.	GROUND 2 BASED ON ALTHOFF IN VIEW OF NYE IN FURTHER VIEW OF SURJANTO IS DEFICIENT (CLAIMS 10, 23, 36).....	56
VIII.	GROUND 3 BASED ON ALTHOFF IN VIEW OF NYE IN FURTHER VIEW OF MOORE IS DEFICIENT (CLAIMS 6-7, 19-20, 32-33).....	56
IX.	SOTERA STIPULATION.....	57
X.	CONCLUSION.....	57

PATENT OWNER'S EXHIBIT LIST

- EX2001 J. Kufner, R. Mařík, “State Machine Abstraction Layer”, 2nd Information and Communication Technology - EurAsia Conference (ICT-EurAsia), Apr 2014, Bali, Indonesia. pp.213-227, 10.1007/978-3-642-55032-4_21, HAL open science, hal-01397199
- EX2002 J. Kufner, R. Mařík: Restful State Machines and SQL Database, IEEE Access, VOLUME 7, 2019, Digital Object Identifier 10.1109/ACCESS.2019.2944807 pp. 144603-17
- EX2003 Finite-state machine – Wikipedia
- EX2004 Siobhan Roberts, “The Lasting Lessons of John Conway’s Game of Life” The New York Times, Dec. 28, 2020
<https://www.nytimes.com/2020/12/28/science/math-conway-game-of-life.html>

I. INTRODUCTION

The Board should deny institution of the Petition for *Inter Partes* Review of U.S. Patent No. 7,979,468 (“the ’468 Patent”) because the ’468 Patent claims a novel and non-obvious arrangement of modules and methods for relational database systems absent from the prior art. The Petition’s attempt to gloss over material gaps in the prior art should be rejected.

The ’468 Patent, entitled “Database Data Dictionary,” concerns an extension to a relational database platform with a unique set of modules that permit database creators to configure a database using a table structure that is independent of the data model of the system and to manage workflows using state machines without needing to write dedicated code. Rather than having to manually define tables and their relations, a database creator enters the database information into predefined modules (table-based structures), leading to an improved experience.

The ’468 Patent explains that in the prior art, a “conventional data storage system implements its own data model according to the user interface and the business rules specification for that system.” ’468 Patent, 1:20-22. “Such systems depend upon a specific data model and lack flexibility and portability.” *Id.*, 1:26-27. For example, if a database creator wished to design a database, the designer would build it from scratch—deciding what tables and relationships among those tables would be optimal. Any changes to the table organization could be highly

problematic and require a reworking of the data model.

The '468 Patent provides a novel and non-obvious modification to prior art relational databases. For example, claim 1 recites a system that contains eight specific database modules: an “element module,” an “element relation module,” a “class module,” an “attribute module,” a “class attribute module,” a “type definition module,” a “state machine module,” and a “status module.” The claim further recites details concerning each of these modules and how they interact. As explained below, the modules comprise database tables or columns from database tables. This same set of modules is used for each database the database designer wishes to design, thus simplifying the database creation process.

Nothing comparable to this is disclosed in the prior art of record.

The Petition goes awry from the first sentence of the '468 Patent Overview, where Petitioner asserts that the patent “is directed to object-oriented (‘OO’) data modeling.” Petition, 1. This is incorrect. The term “object-oriented” does not even appear in the '468 Patent. The Examiner made the same error and issued a rejection based on object-oriented prior art. In response, the Applicant explained that the Examiner was mistaken and that the invention was not drawn to standard object-oriented structures, but rather something different. Ex. 1002, 42-48. The Examiner agreed and issued the claims. *Id.*, 8, 11-13.

Thus, the Petition in essence repeats the same basic argument considered by

the Examiner and overcome during prosecution. Like the Examiner, the Petition attempts to shoehorn the invention into the “object-oriented” database framework where it does not belong. To be clear, the invention is drawn to relational databases—it is based on tables and relations between tables, and the claimed modules are “not drawn to standard object-oriented structures.” Ex. 1002, 47.

This Preliminary Response lays out four independent reasons why this proceeding should not be instituted.¹ The first is procedural and the remaining are substantive.

First, the Petition is deficient because a petition must present arguments based on what the Petitioner contends to be the correct claim construction. Petitioners may not waste the Board’s time with arguments based on a broad claim construction Petitioner believes to be incorrect. Here, the Petition relies on a construction of the term “module” that the Petitioner stated was incorrect in a parallel proceeding at the time it filed the Petition. *Compare* Petition, 4 *with* Ex. 1011, 2. Petitioner further alleges that the construction used in the Petition is

¹ Given the early stage of the proceedings, the word limits, and the need to read the evidence in the light most favorable to the Petitioner, Patent Owner does not provide arguments for each missing limitation, but rather highlights a few deficiencies in the Petition.

allegedly Patent Owner's construction. It is not. It is simply an incorrect claim construction. As such, institution can be denied on this basis alone.

Second, the Petition fails to locate the "class module" of the claims in the prior art. In fact, Althoff generates a relational database that has no need for a "class module" and Althoff in fact does not generate such a class module. The Petition relies on a non-existent "class Class 301 table" and is thus deficient.

Third, even assuming the material from Althoff cited in the Petition constituted a "class module" (it does not), the Petition relies on the wrong database. For the data repository and the element module, among others, Petitioner relies on aspects of the **relational database** of Althoff. But for the "class module," Petitioner relies on material describing the **object database** of Althoff. This is improper because the two databases are (1) different databases and (2) incompatible—and the content of one is **translated** into the other upon a trigger event. Petitioner cannot mix and match aspects of the relational database and Althoff's object database. Thus, even if Petitioner were able to find a "class module" in Althoff's object database (and it does not), the Petition would nonetheless be deficient because the alleged module would be in the wrong place.

Fourth, the Petition fails to locate the "state machine module" and "status module" of the claims in the prior art. Here the Petition brings in the Nye reference. However, Nye does not disclose state machines, but rather discloses an

event model. There is neither a “state machine module” nor a “status module” in Nye.

For each of these additional reasons, institution should be denied.

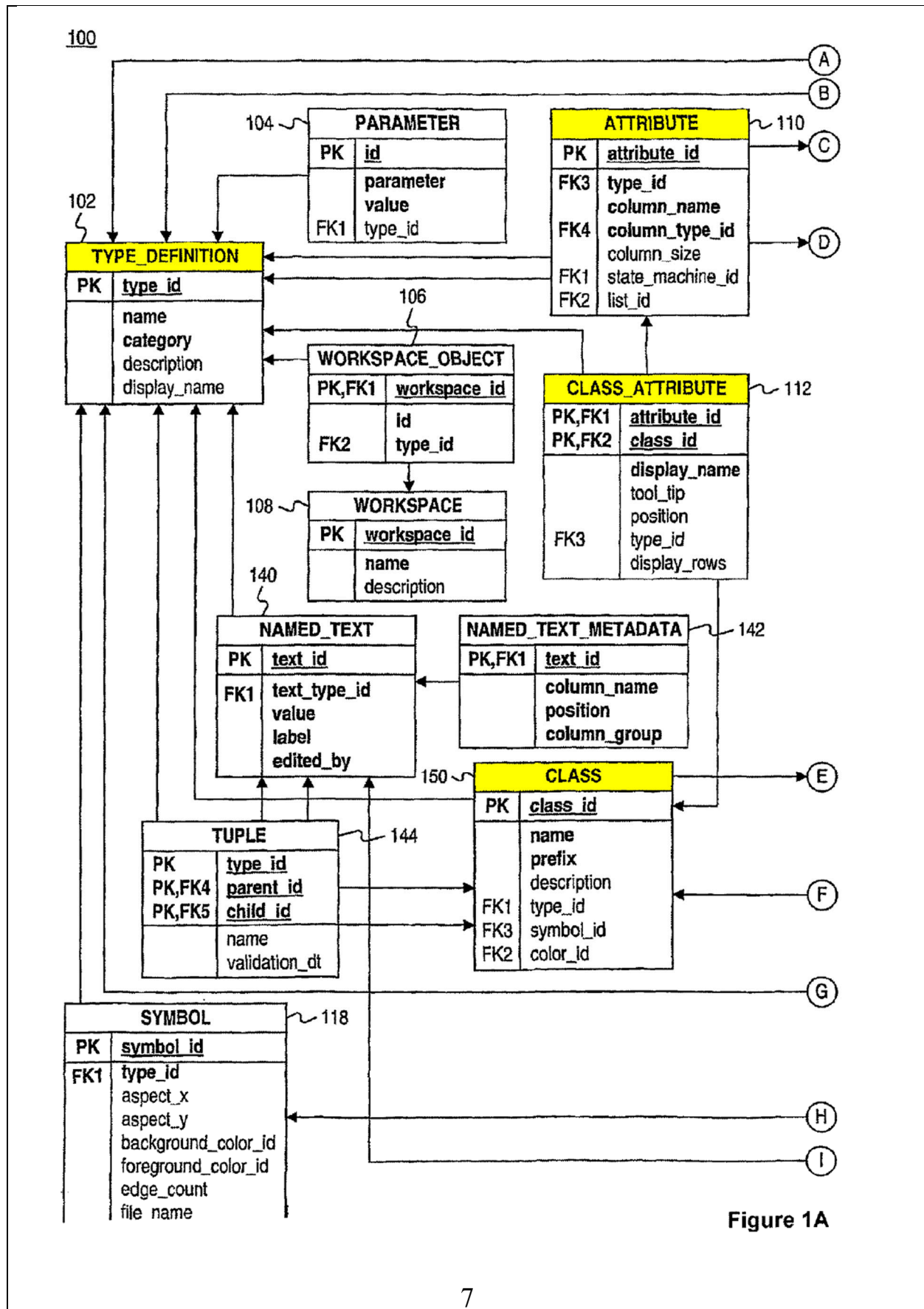
II. SUMMARY OF THE '468 PATENT

The '468 Patent is directed to a novel extension to relational databases. The novel construction and configuration of the '468 Patent concerns the various interrelated modules that work together to provide an improved way to work with data sets and workflows. '468 Patent, 1:65–67. The modules are table-based structures and may be implemented as one or more tables or a subset of the columns thereof. One benefit of the invention is that the user can develop a database structure without concerning his or herself with the data model needed to implement the design of the relational database. Rather, the user just needs to add the data to the pre-defined table structures (the “modules”) defined in the '468 Patent. Another benefit is that a user may easily control dynamic data sets (data that changes based on various conditions) with little or no dedicated code by using state machines within a relational database structure (again, by inserting data into the pre-defined modules). '468 Patent, 1:20-28, 1:45-52, 4:38-41; 7:59-8:18.

A relational database is a type of database that stores and provides access to data, which the database maintains in tables having rows and columns. In a relational database, one or more columns of a table may serve as an index, and the

indices may be used to create relationships among the tables. The specification of the '468 Patent uses relational databases in a new way and discloses an embodiment in which various interrelated modules are implemented as database tables to provide increased flexibility, portability, and ease of use. '468 Patent, 3:1-4, 3:59-5:18.

Figures 1A and 1B illustrate one embodiment of the invention, which contains the data structures corresponding to the eight claimed modules of claim 1 and others too. In the reproduced figures below, the eight modules required by claim 1 (implemented as individual tables in this embodiment) are highlighted:



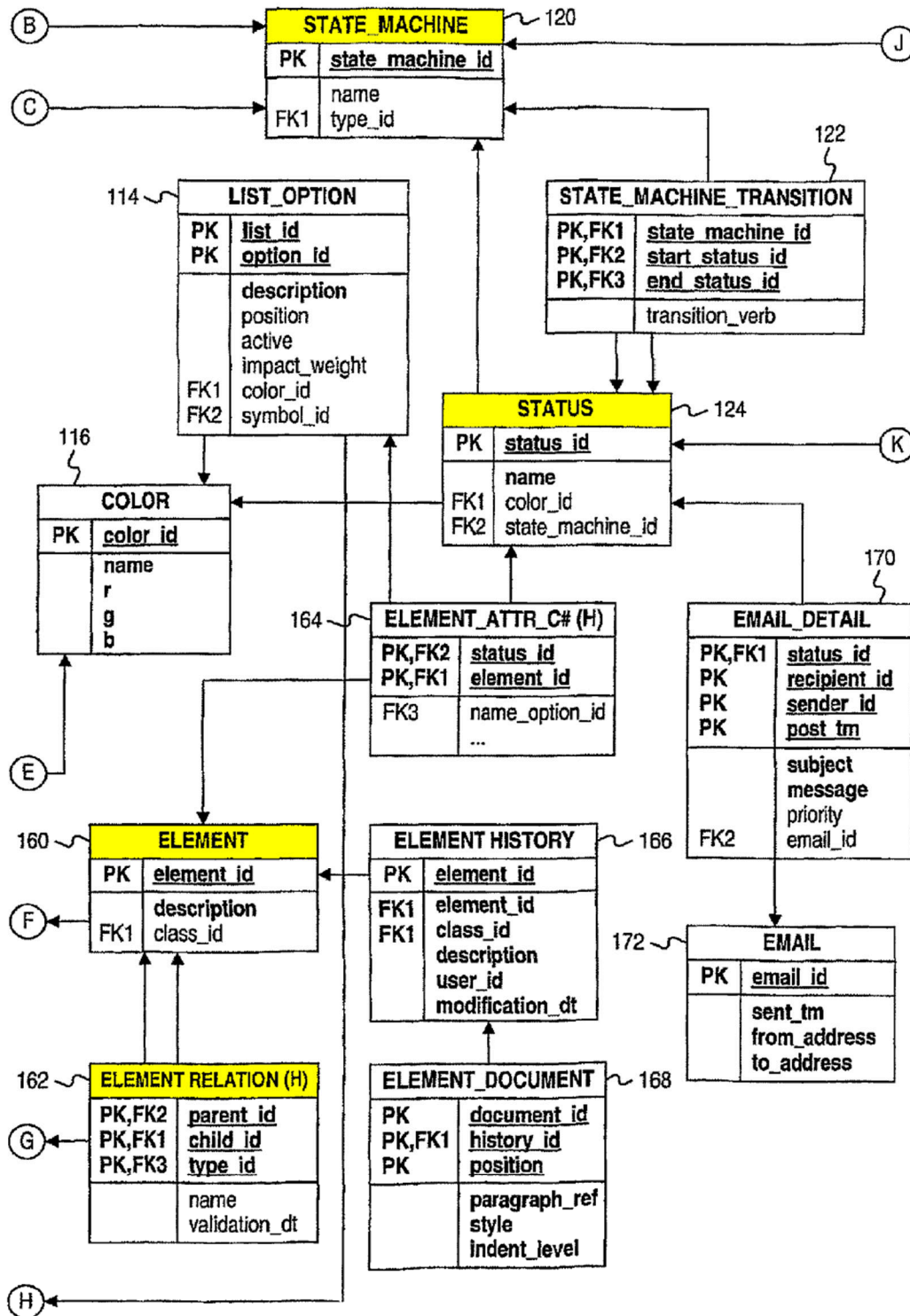


Figure 1B

To provide a concrete example of one possible application of the invention, assume a database designer desired to build a database to track certain employees in an organization, where an employee can have a name, birthday, and social security number. The employees may also have a status of being either a prospective employee, a hired employee, an employee on probation, or an employee terminated for cause. Further assume that employees are assigned to projects and projects have a duration and project lead. Ex. 1001, 4:20-21, 8:4-10; 13:4-21. Prior to the invention of the '468 Patent, that user had various options. For example, that person could maintain one big table in a spreadsheet with this information:

Employee Name	Birthday	SS#	Status	Project	Project Duration	Project Lead
John Doe	1/1/1975	111-22-3333	Hired	Hardware	6 months	Helen Harper
Jane Smith	2/1/1980	444-55-6666	Hired	Software	8 months	Sally Spade
Tina Jones	3/1/1990	777-88-9999	Prospective	Finance	3 months	Frank Foster
Tom Wilson	4/1/1985	222-33-4444	Terminated	Hardware	6 months	Helen Harper

Alternatively, the database designer could create a relational database with multiple tables. One possibility is to generate two tables, one for employee information and one for project information:

Employee Key	Employee Name	Birthday	SS#	Project Key	Status
E1	John Doe	1/1/1975	111-22-3333	P1	Hired
E2	Jane Smith	2/1/1980	444-55-6666	P2	Hired
E3	Tina Jones	3/1/1990	77-888-9999	P3	Prospective
E4	Tom Wilson	4/1/1985	222-333-4444	P1	Terminated

Project Key	Project	Duration	Project Lead
D1	Hardware	6 months	Helen Harper
D2	Software	8 months	Sally Spade
D3	Finance	3 months	Frank Foster

The “foreign key” in the employee table is used to locate the project information by matching the “primary key” in the project table. *See, e.g.*, Ex. 1005:4-17 (discussing how prior art relational databases related two tables). This table structure is relatively straight-forward. However, as databases grow in complexity with many more tables, it can be difficult to define an optimal table structure.

Now assume the database designer wishes to control what can be entered into the “status” column. For example, the database designer may want to ensure that an employee cannot be terminated without first being on probation and cannot be put on probation without being hired. Ex. 1001, 8:4-10. The database designer would either be out of luck or would have to write dedicated code to handle such restrictions. *See, e.g.*, Ex. 1001, 8:50-60.

Instead of this conventional approach, the '468 Patent does not require the database creator to manually design a table structure. Instead, the database creator using the invention would enter the database information into the specified modules. For example, the database creator would decide which of the data items are the items of interest and enter those into the class module. These could be, for example, the employees and projects. If the database creator decided at a later date to track company departments or maybe assigned company cars, those too could be added to the class module.

The database creator would then define the attributes of interest and list those in the attribute module. These might be “employee name,” “project name,” “employee birthday,” and “project duration,” among others. The database creator would add the individual employee and project information into the element module. The database creator would continue this process and enter information into the remaining modules. For example, if employees are to be assigned to projects, then that information would be entered into the element relation module. If the database creator decided at a later date to track employee telephone numbers, employee addresses, or project leads, that information can be added as attributes to the attribute module. '468 Patent, 3:52-5:39.

Next, the user can control the workflow related to employee status without writing dedicated code. Instead, the user defines a state machine in the state

machine module, enters the different allowable states into the status module (prospective, hired, probation, terminated) and the allowable transitions into a transition module (which is not specifically called out in claim 1, but disclosed in the embodiment of Figures 1A & 1B). No dedicated code is required. '468 Patent, FIGS. 1A, 1B, 5, & 6, 7:59-8:40.

A database designer's task is thus simplified and enhanced by using the modules defined in the '468 patent to create the desired database rather than the prior art relational databases.

* * *

The Petition ignores these details and declares that the patent "is directed to object-oriented ('OO') data modeling, a technique that has been known in application software development since the 80s." Petition, 1-2. The Petition misreads the '468 patent, which is directed to relational databases and not object-oriented data modeling. To be sure, the '468 patent uses terms that are also found in object-oriented programming, such as the terms "class" and "attribute." However, the class and attribute modules are not object-oriented structures, but rather are relational database structures. That is, they are table-based structures.

Thus, the '468 Patent uses a combination of relational database modules to simplify the process of designing databases. A user need not manually design a complex table structure. Instead, for example, the user can enter the categories of

information to be tracked into the class module, create attributes and their properties and store that information in the attribute module, and store the individual elements in the element module. The user can further develop state machines to control the behavior of elements in the database and implement them by simply entering the information into additional pre-defined modules, thus obviating the need for custom code to be developed for this function.

III. SUMMARY OF THE CITED REFERENCES

A. Althoff (EX1006)

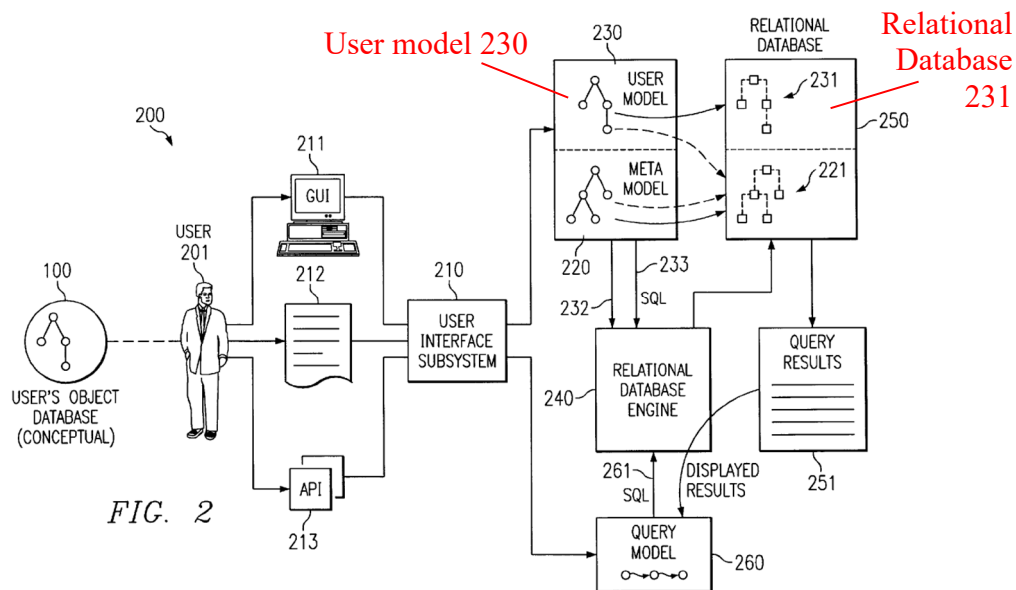
Althoff discloses a method and system for translation between an object-oriented database and a relational database. Thus, Althoff has little relevance to the '468 Patent and is directed to a different problem. Moreover, Althoff's disclosure of the details of its relational database is thin, leaving many gaps that the Petition cannot fill.

Althoff states that “object-oriented database” or “OODB” applications have grown in prominence. However, Althoff explains that “one problem which has arisen in the art is that many database systems have been designed to operate with another, different database technique, known as ‘relational database’ management (RDBM).” Ex. 1006 (Althoff), 1:41-57.² To bridge this chasm,

² Emphasis added throughout unless otherwise noted.

Althoff discloses a database translator. Using the system of Althoff, a user “may create, edit, and manipulate a user’s object database” and the user’s object database is “dynamically translated into a set of relational database structures.” *Id.*, Abstract.

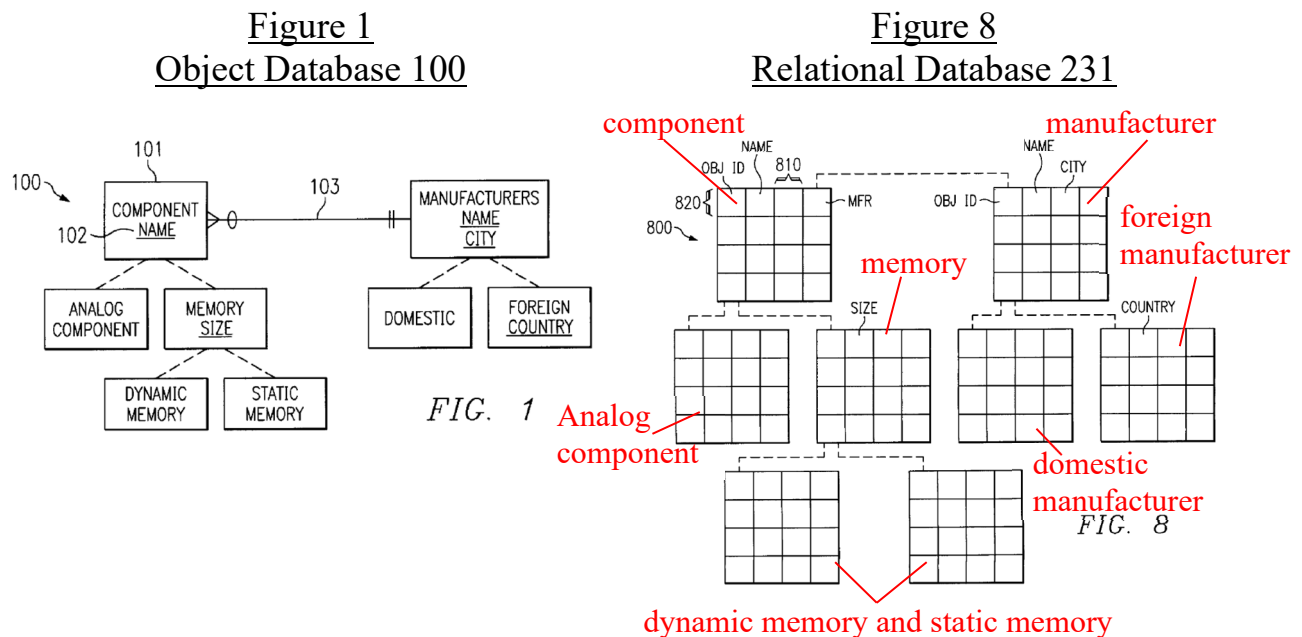
Figure 2 of Althoff clearly distinguishes between the two databases within Althoff’s system. On the one hand there is “user model 230,” which holds the user’s object database. On the other hand, there is the “relational database 231,” which is created by the system via the database translation process.



Ex. 1006, Fig. 2. “In response to a triggering event, the System 200 translates the user model 230 into a user relational database 231, using a set of SQL commands 232 for specifying and creating relational structures in the user relational database 231.” Ex. 1006, 8:18-21.

Althoff provides an example of the translation from one database to another in Figures 1 and 8. Figure 1 depicts an object database stored in an object-oriented model. Each box in Figure 1 represents a class to which objects can belong and the database would be populated with additional objects that would be instances of one of these classes. For example, if “Acme Corporation” was added to the object database as a domestic manufacturer, an object of class “domestic” would be created that holds the relevant information concerning Acme Corporation.

Figure 8 depicts a relational database created by the database translator from the object-oriented database shown in Figure 1. *Id.*, 4:58-59, 5:8-10.



As illustrated above, each class created in the object database has a corresponding table in the relational database. Although Figure 8 does not name

each table, it is possible to identify tables by their columns. For example, the component table has a “name” column which corresponds to the “name” property in the object database and the manufacturer table has “name” and “city” columns which correspond to the “name” and “city” properties in the object database.

The component and manufacturer tables are related using “keys” as was standard in relational databases. The component table has a column to hold the foreign key to rows in the manufacturer table, allowing each component to be associated with a manufacturer. This column is labeled “MFR,” presumably to refer to “manufacturer.” *See* Petition, 5; Ex. 1006, Fig. 8.

Althoff Figure 8 further illustrates via the vertical dotted lines the “inheritance relationships” between related classes. *See* Petition, 5; Ex. 1006, 3:61-63. Althoff explains that an object may be broken apart with pieces stored in different relational tables: “An object member of the derived class is entered in both the table for the base class and the table for the derived class, and it is given the same object-ID in both tables.” Ex. 1006, 3:64-67. For example, a static memory component would have a row in each of the static memory table, the memory table, and the component table. Each table would have a row identified by the same object ID (because each row concerns the same object). Likewise, a domestic manufacturer would have a row in the domestic manufacturer table and a row in the manufacturer table. Each of these two tables would have a row with the

same object ID. *Id.*

To retrieve all the properties related to a static memory component, for example, one must read the rows with the relevant object-ID from multiple tables. That is, one must extract and combine the information from the static memory table, the memory table, and the component table. From those three tables, the static memory object may presumably be recreated (although these details are absent from Althoff).

Taking the earlier example of the “Acme Corporation” object, the object database would contain an object called “Acme Corporation” that would contain all the information relevant to “Acme Corporation.” However, when that object is translated for insertion into the relational database, rows will be generated in the “manufacturer” table, including the name and city for Acme Corporation and additional rows will be generated in the “domestic” table, which would include additional information specific to domestic corporations and not relevant to foreign corporations. *See* Ex. 1006, 3:64-67.

B. Nye (EX1005)

Nye provides “dependency logic” for a database via the building of an “event model” that works “without having to modify the underlying structure of the database.” Ex. 1005, Abstract. The event model contains events, wherein “[e]ach event represents a set of actions that are optionally contingent upon a

condition.” *Id.* The event model is similar to a rules engine, much like that found in Microsoft Outlook, where a user may create rules to process incoming mail. For example, a user may specify that when mail comes in from a certain address, it is forwarded to a specific folder and a visual alert is triggered. Nye discusses a “subscription expired” event that will check each entry in a subscriber database to discover whether the subscription has expired or not. When “it discovers a user whose subscription is expired the event will send that user a newsletter.” Ex. 1005, 11:38-47.

Nye provides an example where users are categorized based on the status of their subscription. As depicted in Figure 8 of Nye, “if, for example, user 801, 803, and 805 sign up to receive a free 6 month subscription to an online newsletter these users belong to the free trial subscriptions category 820.” Ex. 1005, 7:26-28. “If after 6 months passes users 803 and 805 choose to continue receiving the newsletter and begin paying for it, users 803 and 805 are placed in the paying subscribers category 830.” *Id.*, 7:28-32. Then, if “user 805 then decides to cancel the subscription, then user 805 is placed in the canceled subscriptions category 840.” *Id.*, 7:32-34. Nye explains: “Placing users into separate categories, for example, allows separate actions to be performed for each group of users. A database administrator could decide, for example, to send an electronic mail message to the users who are in the paying subscribers category to thank them for

subscribing or to offer additional services.” Ex. 1005, 7:53-57.

Nye thus provides an “event model” to trigger actions based on the status or state of records and events in the database. Nowhere does Nye state or suggest that it employs a state machine or provide the modules claimed in the ’468 patent. In fact, Nye suggests the opposite. For example, Nye teaches that event models can be created “**without having to create new tables** or change the relationship that exists between tables.” Ex. 1005, 6:4-8. In Nye, the event model lives outside the relational database.

IV. CLAIM CONSTRUCTION

A. Module

Contrary to Petitioner’s arguments, the claim term “module” should be treated no differently than the other claim terms. Namely, it should be understood according to its “ordinary and customary meaning as it would have been understood by a POSITA at the time of the alleged invention, in the context of the ’468 Patent and its prosecution history.” *Phillips v. AWH Corp.*, 415 F.3d 1303 (Fed. Cir. 2005) (*en banc*).” See Petition, 4.

In the context of the ’468 Patent, which concerns relational databases, the term “module” would be understood to be a table-based structure. That is, a module comprises one or more columns within one or more tables. Accordingly, Patent Owner submits that no construction is necessary, but to the extent that the

Board issues a formal claim construction (which is not necessary at this stage of the proceedings), Patent Owner offers the following:

The term “module” within the context of the ’468 Patent means “a table or, one or more columns of/within one or more tables.”

B. The Petition Misstates Patent Owner’s Construction of “Module”

Oddly, the Petition does not state what Petitioner contends to be the correct construction of the term “module,” as required by the rules of this proceeding.

Instead, the Petition first presents a circular argument. The Petition asserts that because the ’468 Patent describes a “data repository” that includes “modules,” a “POSITA would thus understand ‘module’ to be a portion of a data repository.” Petition, 4. It is unclear if the Petition is simply stating a tautology or is attempting to provide a definition for the term “module.”

Next, the Petition states that Patent Owner provided a “definition” of the term “module” at a District Court hearing. That is incorrect. The hearing concerned a motion to dismiss and was held many months before the claim construction proceedings commenced. At the hearing, counsel for Patent Owner was discussing “an example that’s used in the specification” and stated that in that example, the specification depicts “a table or a portion of a table.” The discussion was not intended to “define” the term “module.” Ex. 1013, 78.

Subsequently, the parties provided their proposed claim construction for the

term “module.” Patent Owner stated, as it does here, that module should be given its “[p]lain and ordinary meaning.” Ex. 1011 (2023-06-05 Joint Claim Construction Chart), 2. Petitioner sought the following construction: “A table or, one or more columns of/within a single table.” *Id.* Notably, the parties filed the Joint Claim Construction Chart on June 5, 2023, 18 days before Petitioner filed the Corrected Petition in this proceeding on June 23, 2023. And yet the Corrected Petition makes no mention of the parties’ claim construction positions.

Accordingly, the proper construction is “plain and ordinary meaning,” and to the extent that the Board decides to issue a formal construction, it should be “a table or, one or more columns of/within one or more tables.”

However, no construction is necessary in this proceeding because the Petition is deficient even under the construction stated in the Petition.

**V. INSTITUTION SHOULD BE DENIED UNDER
37 C.F.R. § 42.104(B)(3) BECAUSE THE PETITION DOES NOT
APPLY WHAT PETITIONER CONTENDS TO BE THE PROPER
CONSTRUCTION OF THE TERM “MODULE”**

The Petition first states that a person of ordinary skill in the art “would understand ‘module’ to be a portion of a data repository.” Petition, 4. Next, the Petition asserts that Patent Owner defined “module” to mean “a table or a portion of a table” and then states: “Petitioner applies Patent Owner’s own definition of

‘module’ throughout this Petition.” Petition, 4.

Petitioner is, in fact, incorrect, as Patent Owner contends that the term “module” should take on its ordinary meaning in the context of the ’468 Patent, which is a table or, one or more columns of/within one or more tables. This dispute need not be resolved here because the Petition is deficient under either construction, and Patent Owner will use the construction offered in the Petition in this Preliminary Response. *See Realtime Data, LLC v. Iancu*, 912 F.3d 1368, 1375 (Fed. Cir. 2019) (“The Board is required to construe ‘only those terms . . . that are in controversy, and only to the extent necessary to resolve the controversy.’”).

As a threshold matter, however, the Board need not reach the substance of the dispute because the Petition fails to comply with the basic requirements under the procedural rules. The Petition must set forth “how the challenged claim is to be construed” and “how the construed claim is unpatentable.” 37 C.F.R. § 42.104(b)(3)-(4). If the Petition asserts unpatentability solely under a construction the Petitioner believes is incorrect, then the Petition is deficient and it should not be instituted. *Toyota Motor Corp. v. Blitzsafe Tex., LLC*, IPR2016-00422, Paper 12, 26 (P.T.A.B. Jul. 6, 2016) (“The ‘construction’ referred to by 37 C.F.R. § 42.104(b)(3) is the construction proposed by the Petitioner, one that Petitioner **believes is the correct construction** under applicable law and should apply in the involved proceeding”) (denying petition); *see also Carefusion Corp. v. Baxter*

Int'l, Inc., IPR2016-01456, Paper 9, 7 (P.T.A.B. Feb. 6, 2017) (“Petitioner states that it relies on constructions advanced by Patent Owner in the related litigation; remarkably, however, Petitioner also states that it does not agree that those constructions are in fact correct.”) (denying petition).

Here, Petitioner does not assert that the construction of “module” used in the Petition is correct. In fact, Petitioner is clear to point out that the construction it applies is allegedly one offered by Patent Owner, and not one that it accepts as correct. Petition, 4. Although not discussed in the Petition, Exhibit 1011 provides what Patent Owner contends is the correct construction, and that construction is narrower than the construction relied upon in the Petition. In the Petition, Petitioner relies on multiple tables from Althoff (and sometimes no table at all) to identify what it contends is a “module.” Petition, 24, 26-27 (relying on “class Pointer Valid Class 305” as a module), 28 (relying on an “event model” as a module), 29 (relying on a “record state” as a module), 37 (relying on a “database record” as a module), 38 (relying on a “query model” as a module), and 44 (the combination of multiple “relational class tables” is a module). However, in the District Court, Petitioner contended (and continues to contend) that a module must

be of/within a **single** table.³ On the basis of this inconsistency alone, the Petition should be denied.

Indeed, since the revised § 42.100(b) took effect, the Board has repeatedly declined institution where a petitioner, like Petitioner here, maintains inconsistent claim construction positions. *See Orthopediatrics Corp. v. K2M Inc.*, IPR2018-01546, Paper 10, 10-12 (P.T.A.B. Feb. 14, 2019) (“Petitioner’s failure to provide a claim construction is . . . further compounded by the fact that Petitioner takes an inconsistent position before the District Court . . . by failing to reconcile its proffered claim construction here with its very different construction proffered in District Court . . . Petitioner fails to satisfy [its] burden”) (denying institution); *see also Samsung Elecs. Co., Ltd. v. Ancora Techs., Inc.*, IPR2020-01184, Paper 11, 27 (P.T.A.B. Jan. 5, 2021) (“Significantly, Petitioner’s [claim construction] argument . . . seems to be inconsistent with its position advanced in the parallel litigation and the Federal Circuit’s interpretation . . .”) (denying institution). Thus, a petitioner may not take inconsistent claim construction positions between district court and

³ Ex. 1011 contains Petitioner’s construction before the District Court, which is: “a table or, one or more columns of/within a single table.” The Petition does not use this definition, and does not explain why it chose to use a different definition in the Petition. Another distinction is that in the Petition, a module may be any portion of a table whereas in the District Court the portion must be one or more columns.

IPR proceedings without justification, and by failing to address those inconsistencies, “Petitioner fails to satisfy [its] burden” for institution purposes.

Orthopediatrics, IPR2018-01546, Paper 10, 12.

VI. GROUND 1 BASED ON ALTHOFF IN VIEW OF NYE IS DEFICIENT (CLAIMS 1-9, 11-22, 24-35, 37-39)

A. The Petition fails to identify a “class module” or its methods and, at the very least, fails to identify a “class module” in the same “data repository” as the “element module” (Limitations 1[e], 14[c], and 27[c])

1. Summary

The claims require a “data repository” that has at least eight modules, two of which are an “element module” and a “class module.” The Petition relies on **two different and incompatible databases** in Althoff to find alleged teachings of these modules. The object database is objected-based whereas the relational database is table-based.

First, the object-oriented class description on which Petitioner relies for the “class module” concerns Althoff’s **object-oriented database**, and is not in Althoff’s relational database. However, this is incorrect because (a) the “modules” of claim 1 are table-based structures, which are found in a relational database, and not found in Althoff’s object database and (b) the Petition relies on the relational database of Althoff for the alleged “data repository” that holds the modules. The two databases are distinct, and Althoff specifically provides a translator to

translate an object database into a relational database **because** the object database is not compatible with relational databases. Ex. 1006, 3:4-24; *see Orthopedic Equip. Co. v. United States*, 702 F.2d 1005, 1013 (Fed. Cir. 1983) (nonobviousness considers whether there is a “mismatch” between the technologies being combined or if “there was some technological incompatibility that prevented their combination.”).

Second, Althoff’s relational database structure obviates the need for a “class module,” which explains its absence from Althoff’s relational database. Specifically, Althoff’s relational database preserves the class information after translation by creating separate tables for each class created by the user (which Petitioner maps to the “element modules”). Thus, the Althoff preserves class information within the alleged “element modules” of the relational database. Within the context of Althoff, a “class module” would serve no purpose.

Third, the text on which Petitioner relies for the alleged “class module” in Althoff is not a table or table-based structure, but a description of an object-oriented class. Accordingly, under the construction of “module” applied by the Petition (or under the correct construction), the Petition fails to identify a “class module.”

2. The Petition maps the “data repository” to Althoff’s relational database

The Petition maps the “data repository” of claim 1 onto the relational database of Althoff:

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*.

Petition, 14 (italics emphasis in original and underlined emphasis added).

The Petition further states that:

Althoff’s **relational database** stores data contained in the data sets by translating the user’s object database into a set of relational database structures.

Id.

3. The Petition maps the “element module” of claim 1 to Althoff’s tables in the relational database

For the “element module,” the Petition identifies the tables in Althoff’s relational database. The Petition states: “Althoff teaches representing classes and objects from an object-oriented database within a relational database by explaining that ‘each class . . . is modeled by a relational database table . . . and each object that class is modeled by a row in that table.’ Ex. 1006, 3:45-46.” Petition, 16 (also

citing Althoff, 23:10-12). The Petition further states: “Althoff’s **individual class tables are element modules** because each row of the table stores objects (i.e., elements).” Petition, 16.

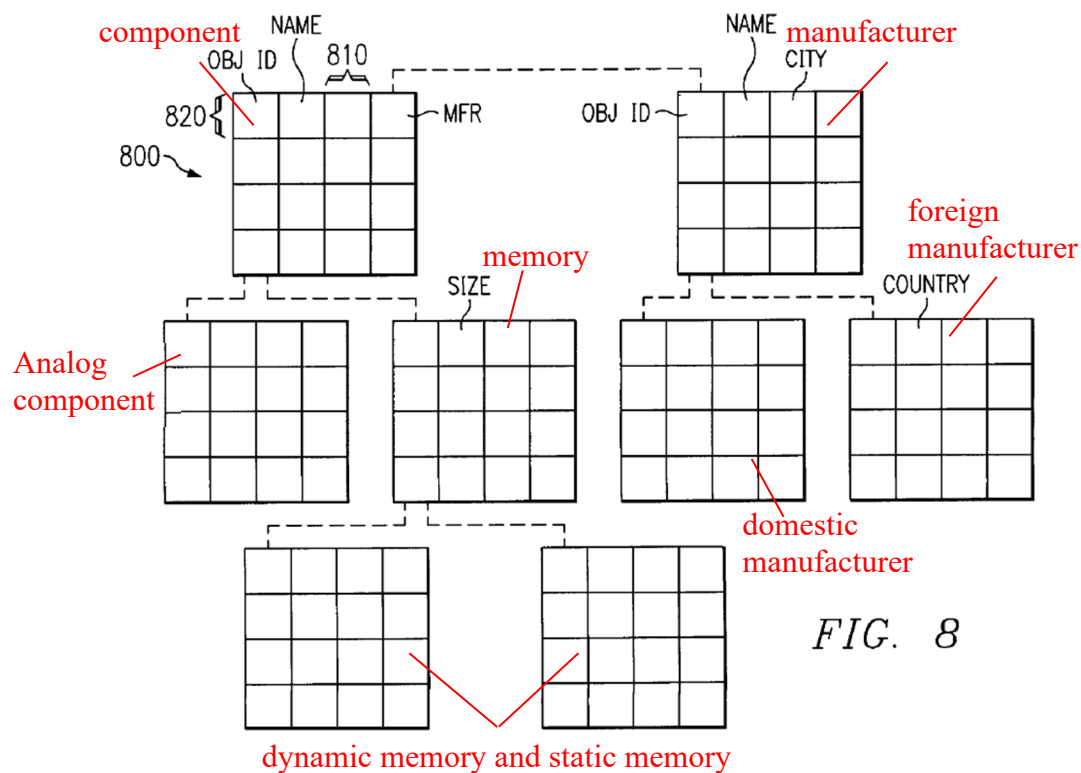


FIG. 8

Althoff, FIG. 8 (annotated to label the tables).

The Petition further states: “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 8 is an example of those tables.” Petition, 17 (discussing the “element relation module”).

In sum, the Petition clearly maps the “data repository” of claim 1 onto the relational database of Althoff and maps the “element module” of claim 1 onto each

of the tables of the relational database as depicted in Figure 8 of Althoff. Even assuming this mapping is valid, the Petition nonetheless fails because the Petition fails to identify a “class module” **in the relational database** of Althoff (*i.e.*, in the alleged “data repository” of Althoff).

4. The Petition maps the “class module” to a class in Althoff’s object database, rather than to a structure in Althoff’s relational database

For the alleged “class module,” the Petition relies on the discussion of the object-oriented class called “Class” that is used in Althoff’s object database prior to the translation of the object database into a relational database. Petition, 21 (referring to “class Class 301”). This class is not a “class module” because it is not a table-based structure—it is an aspect of Althoff’s object-oriented database.

Nor does “class Class 301” interact with the relational database of Althoff, and the Petition does not allege otherwise. On the contrary, Althoff provides a database translator precisely because object databases and relational databases are incompatible: “One problem which has arisen in the art is that the tools for creating, implementing, and manipulating object oriented databases are generally inconsistent with the tools for creating, implementing, and manipulating relational databases” and “[a]ccordingly, it would be advantageous to provide a method of compilation and translation between object oriented and relational database structures.” Ex. 1006, 2:9-14, 3:4-6.

In Althoff, “class Class 301” is the name of an object-oriented class used by the **object database** of Althoff. Ex.1006, 11:12-15. A class in Althoff is not a structure or a thing, but rather is a data type, much like “integer” and “floating point.” Althoff, 1:21-39. Thus, just like the number “3” is a member of the “integer” data type, and “3.33” is a member of the “floating point” data type, a user may create a more complex data type, such as a class called “telephone number” that would allow storage and manipulation of telephone numbers. *Id.* For example, one could envision a system that provides for storage, editing, and dialing of telephone numbers.

In Althoff, when a user wants to create a new class, the user creates an object that is a member of a built-in class (a built-in datatype) called “Class” (Althoff uses the nomenclature “object of class Class.”). The system thereby knows that this new “class” object will not hold data, but rather defines a new class (a new datatype that can be used by the user’s object database). Ex. 1006, 21:28-47. For example, a user may create a telephone number class by creating an object of class Class 301 called “Telephone Number.” That is, the object “Telephone Number” is a member of the class called “Class.”

To be clear, this “Telephone Number” object of class “Class” does not hold any telephone numbers. Rather, it just defines what a telephone number is and what can be done with it. That is, it defines a new data type, and nothing more. If

a user wants to store a telephone number in the database, the user would create a new object such as “Alice’s Number” of class Telephone Number.

In the exemplary object database in Althoff as shown in Figure 1 of Althoff, a user has created 8 classes (data types): (1) Component, (2) Analog Component, (3) Memory, (4) Dynamic Memory, (5) Static Memory, (6) Manufacturer, (7) Domestic [Manufacturer], and (8) Foreign [Manufacturer]:

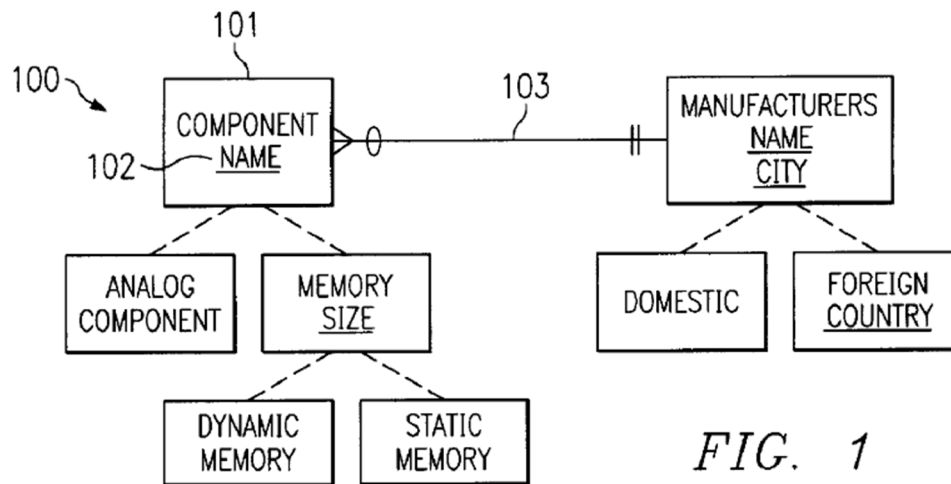


FIG. 1

Althoff, FIG. 1.

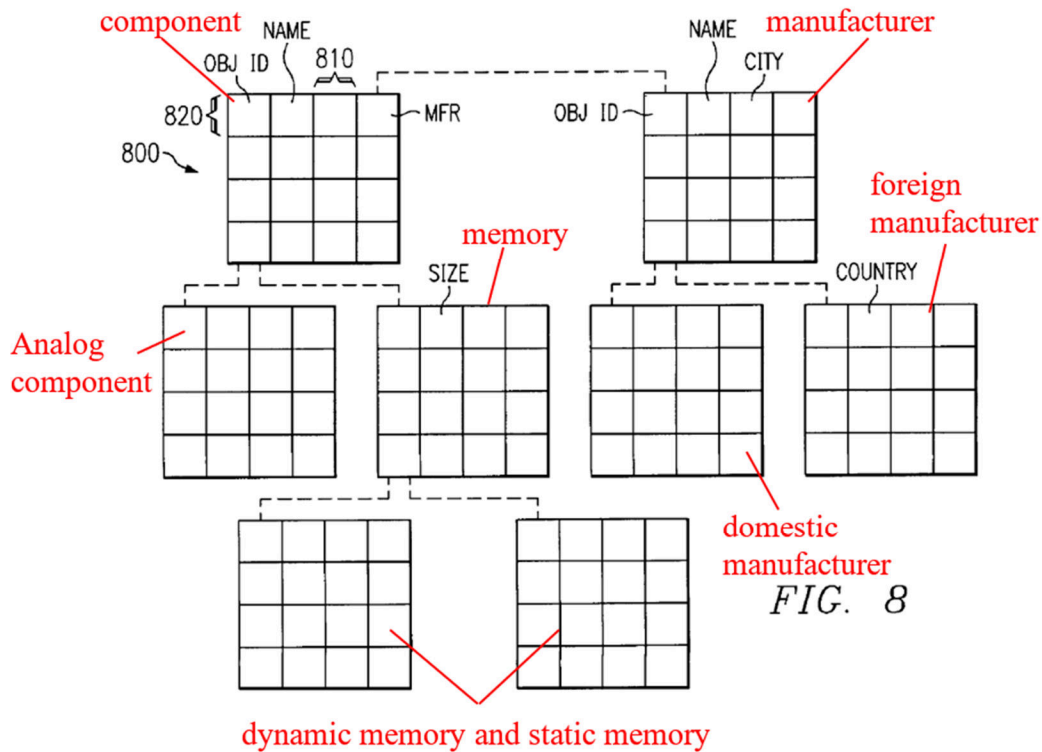
Each of these classes is created in Althoff’s object database by creating an object of class (i.e. of type) Class 301. For example, the “Component” object defines what a component is and what can be done with it, and the “Manufacturer” object defines what a Manufacturer is and what can be done with it.

With this background, the Petition’s mapping of “class Class 301” fails for multiple reasons.

First, “class Class 301” is in the **wrong database**. It is defined in the **object** meta-model and is used by the user’s **object-oriented database** as a datatype. Ex. 1006, 1:28-35, 21:39-47. Each object shown in Figure 1 of Althoff is of type “Class 301.” Ex. 1006, 11:12-14. That is, “Component” is an object of type “Class” and “Memory” is also an object of type “Class.” The same is true of each box in Figure 1. Althoff then translates the object database into a relational database. After translation, each of the classes shown in Figure 1 is translated into a table as shown in Figure 8. Ex. 1006, 23:62-24:8. Petitioner maps each of these tables onto the “element module” of the claims. Petition, 16-17 (stating that “Althoff’s individual class tables are element modules”).

Critically, there is no object named “Class” or “Class 301” in the user’s object database, and no object name “Class” is translated into a “class module” in the relational database. To be clear, Petitioner points to nothing from either the user’s object database or the meta-model that is translated into a “class module” found in the user’s relational database. Ex. 1006, Figs 1 and 8.

In Althoff, the classes in the user’s object database (objects of type “Class”) are converted to “element modules” under the Petition’s mapping (Petition, 16-17):



Althoff creates a “Component” table, “Memory” table, “Manufacturer” table, etc.

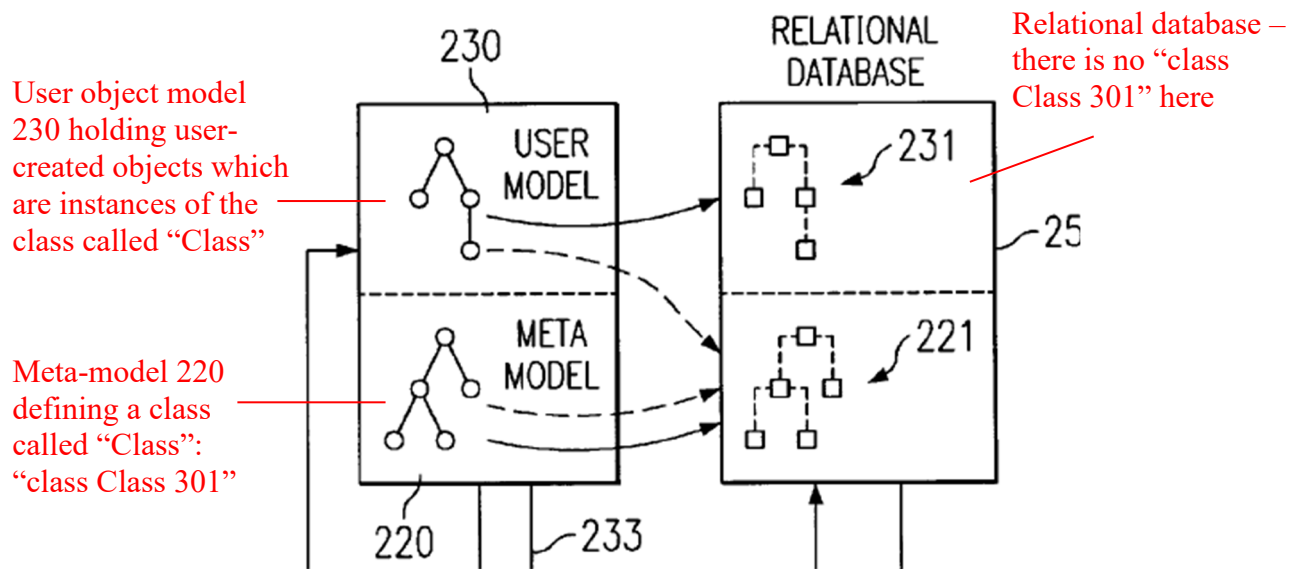
However, no “Class” table is created.

In fact, given the design of Althoff, such as class table (or class module) would be unnecessary. Althoff’s relational database preserves the class information by creating separate tables for each class. Althoff thus has no need for a “class module” in the relational database to keep track of what classes existed in the object database. A “class module” in Althoff would be superfluous.

Second, even if the Petition could combine structures in the object database with structures from the relational database (it cannot), the Petition is still deficient because the “class Class 301” is not structural and thus cannot be a “class module.”

In Althoff, “class Class 301” is a data type defined by Althoff’s meta-model. Ex. 1006, 1:27-31, 11:12-14. As shown in Figure 1, the user creates classes by creating objects that are of **the type** “Class 301,” such as “Component” or “Manufacturer.” In other words, in the Althoff system, a user defines classes by created objects of the type “Class.” Each class is defined in a separate object as graphically depicted in Fig. 1 of Althoff. There is no “object” called “Class” in the user’s object database (or “class Class” or “class Class 301,” or any other variation).

As Althoff explains, “each class of the user’s object database 100 is modeled by an object of class Class 301” as shown in Figure 1. Ex. 1006, 11:12-14. If “the user 201 chooses to build a new class 101, the system 200 proceeds to the step 614” and “[a]t the step 614, the system 200 creates a new object of class Class 301 in the meta-model 220.” Ex. 1006, 21:33-40.



Ex. 1005, FIG. 2 (excerpt). In the portion of Figure 2 shown above, Althoff graphically illustrates the distinct components in the system. User model 230, which holds the user’s object database 100, is on the left. The definition of “class Class 301” is in meta-model 220 and the user model 230 can use “class Class 301” as a data type. Relational database 231 is on the right. The arrow between the two illustrates the translation of one to the other.⁴

⁴ Although Petitioner does not rely on column 7 of Althoff, the Board may note that this column states that the “meta-model” is also represented by a different relational database—“relational database 221.” Ex. 1006, 7:58-62. No information is provided as to how the “meta-model” is stored therein, and there is no suggestion in Althoff that the definition of “class Class 301” is stored as a module (as opposed to, for example, a record in a table, outside of any table, or

5. There is no “Class 301 table” in Althoff

The Petition incorrectly asserts that Althoff discloses a “class Class 301 **table**” that is allegedly a “class module.” Petition, 21 (internal quotations omitted). There is no such table. And there is certainly no such table in relational database 231 that corresponds to the user’s object database. Althoff explains that “class Class 301” is a class in the meta-model to which all user-defined classes in the user’s object database belong: “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. In other words, the user classes “Component” and “Manufacturer” are objects of the type “Class” (they define what a “component” and “manufacturer” are, respectively). Each of these user-defined classes will be translated to a table in the relational database. Ex. 1006, 22:63-24:31. However, there is no “class Class 301” **table**—either in the object database or in the relational database.

The Petition also **incorrectly asserts** that “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

some other format). Moreover, regardless of how it is stored, the definition of “Class 301,” would have no information about the classes shown in Figure. 1. That information would be in the objects within the user’s database. Finally, relational database 221 is a different database than relational database 231.

class Class 301 comprises information in table 3-2.” Petition, 21. This statement misrepresents Althoff because there is no such “table” in Althoff. The Petition partially quotes Althoff, 11:12-16 in support, but the Petition radically alters the meaning of the quoted passage via its partial quotation, ellipsis, and insertion.

What Althoff actually says is:

The user’s object database 100 comprises a set of classes 101. In the meta-model 220, each class of the user’s object database 100 is modeled by an object of class Class 301. In a preferred embodiment, each object of class Class 301 comprises information in table 3-2.

Ex. 1006, 11:12-16. Thus, as the first sentence makes clear, the discussion here concerns the user’s **object database** and not the relational database. The second sentence begins by referring to the “**meta-model 220**”—again, part of the object database, and not the relational database—a detail excised by the Petition’s partial quoting. The second sentence further states that object database classes are “modeled by an object of class Class 301.” Nothing in this sentence states that a “class Class 301” table is created. In fact, Althoff says something quite different. Each user-created class is represented as an object of the class (*i.e.*, of the type) “Class”. Thus, the user creates “Component” and “Manufacturer” objects **of the type** “Class,” but the user’s object database contains no “Class” object, and after translation, there is no “Class” table in the user’s relational database. *See* Ex.

1006, FIG. 1; 5:33-36.

Moreover, Althoff nowhere states that class information for each user created object is stored in a “row” of an alleged “class Class 301 Table” as asserted in the Petition. Petition, 21. Rather, Althoff teaches that distinct objects are created in the object database, and the objects are members of the object-oriented class called “Class.” The objects would contain the information specified by the definition for class “Class.” There is no suggestion in Althoff that the objects are all stored in a table called “Class.” Indeed, if it were true that the classes of the object database (such as “components” and “manufacturers”) were stored “rows” in a “table,” then Althoff would have no need to describe a translation process to convert objects into tables.

Finally, the last sentence partially quoted in the Petition at Page 21 states that preferably the class objects created by the user will have the information recited in Table 3-2. Thus, for example, the “Component” object will include the following information (found in Table 3-2): (1) an Object ID, (2) a Class Label (the class name in the object database, such as “Component”), (3) a Class DB Name (the name to be used when translated into a relational database table, which may also be the Class Label such as “Component”), (3) Number of Objects (in this example, the number of components that have been created), *etc.* Ex. 1006, 11:15-47.

It is important to note that Table 3-2 is merely a descriptive device in the specification of Althoff that describes in a two-column format the information stored within an object of type “Class” in the user’s object database. Nowhere does Althoff state that the information in Table 3-2 is stored as a class table or as a row in a “class table” in the object database.⁵ The tables in the written description simply present information in a form more easily accessible to the reader than a narrative.

B. The Petition fails to identify the “state machine module” and “status module” or their methods (Limitations 1[i], 1[j], 14[g], 14[h], 27[g], 27[h])

1. The claimed invention requires two modules that incorporate state machines into the database

The Petition acknowledges that Althoff does not disclose the “state machine module” and “status module.” For these limitations, the Petition relies on Nye. Petition, 10, 28. However, Nye does not disclose these modules either.

As the Petition acknowledges, a “state machine” is a known prior art concept

⁵ Objects can be complex data structures, as explained by Petitioner’s expert. In paragraphs 35-39 of Exhibit 1003, Petitioner’s expert provides examples of objects. Nowhere in that description does he suggest that objects are stored in tables. On the contrary he discusses **translating** such objects into tables in paragraph 40.

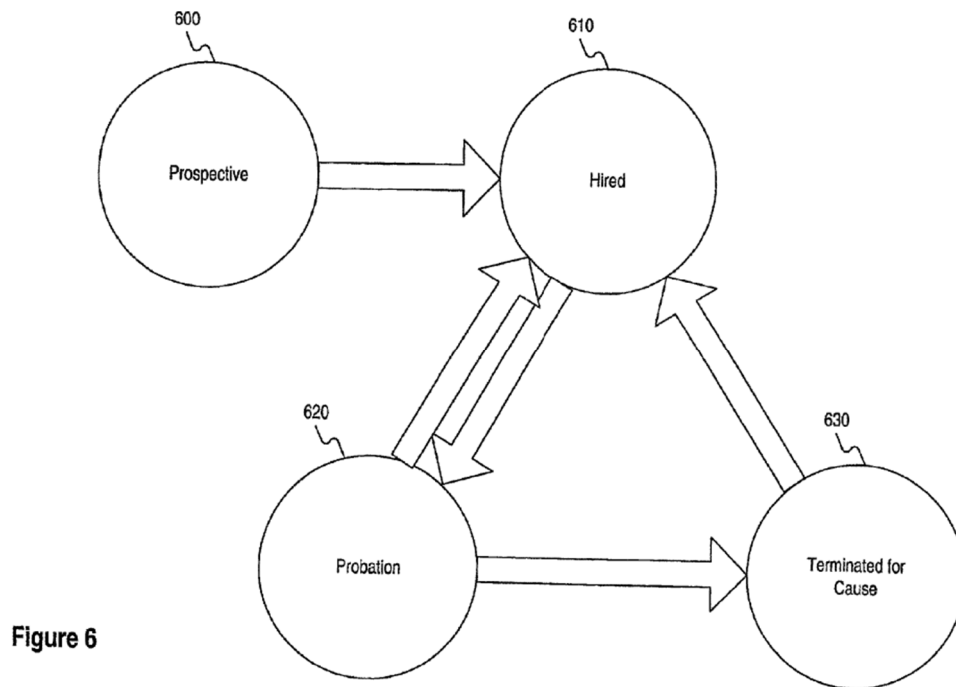
in software development and hardwired logic circuits: “state machines were long-known and common in the field of software development.” Petition, 11. As the petition further acknowledges, a state machine provides “a process where an object has a defined state, and transitions between states” based on a “State Pattern,” as “was common practice in the industry at least by the 1990s.”⁶ *Id*; *see also* Ex. 2003 (Wikipedia) (“a state machine[] is a mathematical model of computation. It is an abstract machine that can be in exactly one of a finite number of states at any given time. The FSM can change from one state to another in response to some inputs; the change from one state to another is called a transition.”).

Thus, the Petition acknowledges that state machines were both old and well-known. It is thus significant that **neither** Althoff **nor** Nye uses the term “state machine.”

In contrast to the database prior art cited by Petitioner, the ’468 Patent uses the term of art “state machine” and incorporates state machines into its database structure. The ’468 patent explains that the invention allows for the definition of a

⁶ Two points of clarification. First, the use of the term “object” here is used in the generic sense and should not be confused with use of the term “object” in the “object-oriented programming” context. State machines predate object-oriented programming. Second, the “industry” being referenced is not the database industry, but the software development industry.

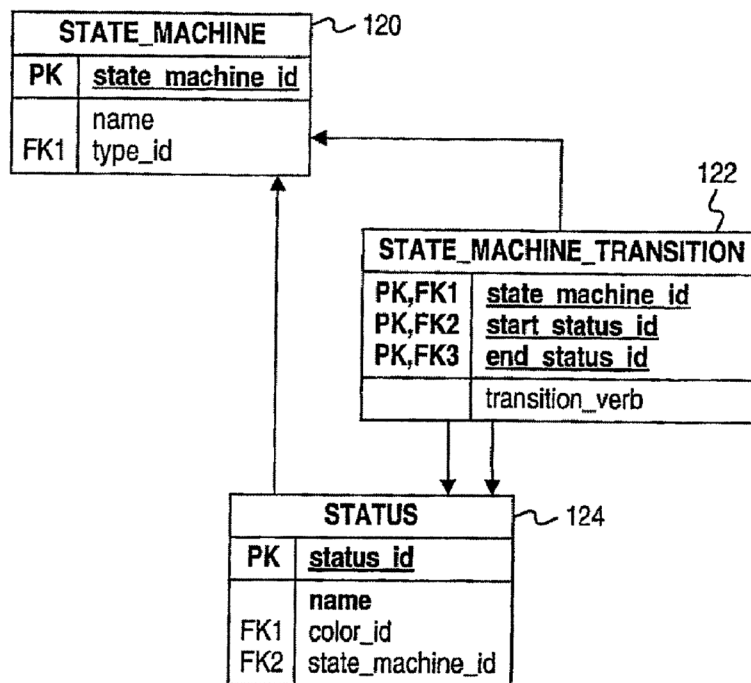
“state machine type which is not found in a relational database.” ’468 Patent, 4:37-39. The addition of a state machine type provides “a very powerful extension of normal relational database attributes or column types.” *Id.*, 7:59-60. The ’468 Patent provides an example of a state machine in Figure 6:



In this example, an employee is within exactly one of four defined states. The employee is either a prospective employee, a hired employee, an employee on probation, or an employee terminated for cause. State transitions are also defined. Most notably, an employee can reach the “hired” state from any of the three other states, but an employee may only reach the state “terminated for cause” from the “probation” state. ’468 Patent, 8:19-40. An employee cannot be in the

“prospective” and “hired” states simultaneously and nor can the employee be in the “hired” and “probation” states simultaneously. The employee element would move from one state to another based on the transition rules specified for the state machine. *Id.*

In one embodiment of the '468 Patent, the state machine is implemented, at least in part, using three tables: state_machine table 120, state_machine_transition table 122, and status table 124. '468 Patent, FIG. 5.



Id.

2. Nye does not disclose a state machine

Nye does not disclose a state machine. In fact, Nye does not even use the

term “state machine.” In addition, the state information of the “event model” of Nye are not stored in the tables of a relational database. Ex. 1005 (Nye), 6:1-8. Thus, Nye cannot disclose any of the state machine module, status module, or the state machine transition module because the “event model” is outside the relational or object database with which it is used.

Instead, Nye discloses a “method and apparatus for event modeling.” Ex. 1005, Title. Rather than have defined states, Nye handles record changes “by using an event model” that has defined events. *Id.* Nye explains: “Each event represents a set of actions that are optionally contingent upon a condition.” *Id.* An action is different than a state. In fact, Nye explains that an event has various attributes including (1) “an internal state,” (2) “[a] way to decide when to try to generate a new internal state,” and (3) “[a] way to generate a new internal state.” Ex. 1005, 7:66-8:8. Thus, Nye treats events and states as different things. An event has multiple “internal states” as attributes and the event may generate new states for itself. Event attributes can also represent things other than states:

There are a variety of different things an attribute can represent, for example, an attribute could represent an internal event state 310, data specific to a particular event 311, data shared by all events having the same event type 312, a way to decide when to try to generate a new internal state 313, and a way to generate a new internal state 314. More than one attribute can

be created and an attributes can represent anything the programmer defines it to represent.

Ex. 1005 (Nye), 12:30-38. Nye further describes a “trigger function [that] indicates whether an event has occurred or not. It does this by taking as input the set of events and their respective states upon which the event having the trigger function depends.” *Id.*, 13:15-18.

Also, a state machine, like that depicted in Figure 6 of the '468 Patent, has nodes that represent states. In stark contrast to a state machine, the nodes in Nye depict events. In Nye, “events in an event metamodel M are represented as nodes of a directed graph G.” Ex. 1005, 7:65-66. Thus, whereas a state machine and an event model are capable of handling active data and may be triggered by events, they are different and operate in different ways. Nye’s nodes represent actions, not states. *Id.*

Moreover, in a state machine, an element is in exactly one state at any given time. Petition, 11-12 (In the “state machine model” an “object has a defined state”), Ex. 2003 (Wikipedia), '468 Patent, FIG. 6. However, in Nye, the events and records may be in multiple states at one time. When describing “state transitions,” Nye uses a subscription example. Nye states that a subscription customer may have different statuses, including “new customer,” “a customer who’s [sic] subscription is about to expire,” “a customer who is a target for a

specific renewal offer,” “a new renewed customer that has not yet paid the subscription fee,” among others. Ex. 1005, 6:59-7:1. Thus, for example, a customer may be both a customer whose subscription is about to expire and a customer who is the target of a specific renewal offer. Such overlapping states would not be present in a state machine, but are permitted in Nye’s event model. Thus, whereas state machines (such as in Figure 6 of the ’468 Patent) in the Petition’s words “are a very clear way to describe behavior of a system” (Petition, 11), the complex event-based system of Nye is the opposite.

Nye provides another example of a newsletter that also has overlapping user states. A user could be categorized as having a “free trial subscription,” “paying subscriber,” or “canceled subscription,” and each category “may have subcategories.” Ex. 1005, 7:10-38. “The paid subscriptions category 820, for example, could contain a sub-category 821 called ‘discount subscribers’ and another subcategory 822 called ‘preferred subscribers.’” *Id.*, 7:39-41.

Nye teaches that the different subscribers can be divided into separate buckets based on the various categories. “Categorizing the database according to the status of the data held in it allows separate actions to be performed on each category. Placing users into separate categories, for example, allows separate actions to be performed for each group of users.” *Id.*, 7:51-55. Thus, the states in this embodiment of Nye are used for categorization, and not for the purpose of

driving a state machine.

Moreover, given that state machines were well-known at the time of Nye, one would expect Nye to use the term “state machine” if it meant to describe a state machine. It does not. Compare the conspicuous absence of the term “state machine” in Nye with its use in articles that **post-date** the filing of the ’468 Patent by a decade or more. Ex. 2001 (*State Machine Abstraction Layer*) (“Smalldb brings a bit forgotten art of state machines into the web development”) (Introduction), Ex. 2002 (*Restful State Machines and SQL Database*) (“State machines and a relational database may look like completely unrelated tools, yet they form an interesting couple.”) (Abstract).

Nye does not teach a state machine, and the combination of Althoff with Nye’s event model (as argued in the Petition) would not result in a database with a state machine, let alone a state machine module and a status module as claimed.

3. The citations to Nye in the Petition offer no support for Petitioner’s assertions

The Petition first argues that “Nye teaches that the elements, referred to as records in a database may be updated and that the ‘status of records in a database changes over time.’” Petition, 28 (emphasis omitted). This vacuous statement does not help Petitioner. Database records often change, and when a record changes, it can be said that the record’s “status” or “state” has changed. For

example, a database record for a component may be updated with a new price or a new address. Such a change of “state” says nothing about whether or not the database includes and accommodates **state machines**.

The Petition next quotes an example where “**a set of events**” is modeled such that “**event** states” are defined. Petition, 28. However, the passage states that transitions in the event model involves interactions among multiple events. Ex. 1005, 12:39-55. Thus, the Petition’s statement that the event model specifies the “various event states **and the transitions between those states**” is unsupported and contrary to the cited passage. *Compare* Petition, 28 with Ex. 1005, 12:39-63. In fact, the transitions are based on the complex interactions of potentially numerous events, and are not specified in advance. As Nye states later in the cited passage, in one exemplary event model, events generate a function that “corresponds to the cell transition rules for John H. Conway’s game of LIFE.” Ex. 1005, 12:59-61. In the game of LIFE, the states are not defined, only a few transition rules are defined that apply to any state. The game of LIFE demonstrates “extremely simple rules can give rise to the most complex and most unpredictable behavior possible.” Ex. 2004 (New York Times). Thus, Nye teaches the opposite of a state machine, which offers predictable results, and Nye’s teachings would not suggest to a skilled artisan to include a state machine within a database structure.

The Petition next states that “Nye also teaches storing one or more state

machine types.” Petition, 29 (emphasis removed). This is incorrect and unsupported. The Petition relies on the following statement in the Abstract of Nye: “An event can represent anything it is defined to represent.” Ex. 1005, Abstract. This passage discusses events, and not states, and this passage does not discuss state machines, or state machine types.

Regarding the status module, the Petition again refers to the description in Nye of records having states. The Petition quotes Nye as disclosing that “[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 record’s ‘state’.” Petition, 29 (quoting Ex. 1005, 7:10-13). It is not disputed that database records can be thought of as having states, and those states can be few (newest or not) or many (the possible states of a monetary amount may be in the trillions). But it is not enough to find a database with states (Althoff’s database has states, and yet Petitioner does not allege that Althoff includes a **state machine**). The question is whether the database implements a state machine, and more precisely whether it implements such a state machine using the claimed state machine and status modules. Neither Althoff nor Nye disclose such a state machine or structure.

In fact, the Petition ignores Nye, Figure 5 in its analysis of claim 1. That figure is described as illustrating “how one embodiment of the invention handles

state transitions.” Ex. 1005, 6:19-20. If Nye disclosed a state machine, one would expect figure 5 to disclose a state machine—but it does not. *Compare* ’468 Patent, FIG. 6 (a state machine) *with* Ex. 1005 (Nye), FIG. 5 (events, event types, and event dependency relationships). As Nye states, each event in Figure 5 can “represent a number of things” such as “the users in a database” or “the number of people who chose to visit a certain web page.” Ex. 1005, 11:1-6. They do not represent states.

Next, the Petition refers to the categorization of records according to record state in Nye. Petition, 30 (discussing Nye, FIG. 8). Figure 8 simply sorts users according to their status in the database. “In one embodiment of the invention the data contained in a database is categorized according to its status. Referring now to FIG. 8 for example, a database of users 800 can be categorized according to the users 801-806 current status.” Ex. 1005, 7:22-25. Thus, users are placed into buckets depending on their status: whether they have a trial subscription, are paying subscribers, or have canceled subscriptions. *Id.*, 7:25-36. Nothing in Figure 8 suggests creating a state machine and storing the state machine using a state machine module and a status module. This is confirmed by reviewing the annotated figure on page 31 of the Petition. In Petitioner’s annotations, Petitioner acknowledges that the figure simply places users into three different categories based on the status of their subscription. No state machine is identified, no state

machine module is identified, and no status module is identified.

C. Dependent Claims

The dependent claims are deficient for the same reasons as the independent claims. The Petition makes incorrect arguments in defense of the dependent claims that are similar to those made for the independent claims. Because there is no need to reach the dependent claims given the deficiencies exposed for the independent claims, this Preliminary Response does not address the dependent claims individually. But that silence should not be read as any sort of agreement with the arguments presented for the dependent claims.

For example, for claim 2, the Petition alleges that an “event_dependency table” in Nye discloses a state machine transition module. Petition, 32. It does not. In fact, there is no “event_dependency table” in Nye and no citation is given for this quotation. Rather, it appears that the Petition is referring to “event_dependency 617” which is “an event dependency value [that] represents a type containment relationship. For example, the event_dependency 617 value describes the set of events that another event depends on.” Ex. 1005, 14:17-21. Thus, the “event_dependency” in Nye is a “value” and is not described as a “table.” The Petition points to Figure 4 of Nye. Petition, 32-33. However, Nye does not describe Figure 4 as a collection of tables. Rather, Figure 4 “illustrates how certain values are interrelated.” Ex. 1005, 6:18. The Board may compare

Figure 4 with Figure 1. Figure 1 is clearly a table and is described by Nye as a “database table that contains customer information.” Ex. 1005, 6:11-12.

Moreover, Nye does not suggest “event_dependency” value stores transitions between states or statues, but rather speaks in terms of events, which are “a set of actions.” Petition, 32.

Similarly, for claim 4, the Petition relies on Althoff for a “type definition module” that defines a “state machine type.” Petition, 35-36. The Petition’s citations do not support this statement. Moreover, for claim 1, the Petition does not suggest that Althoff discloses a state machine. Thus, the Petition’s statement that Althoff discloses a “state machine type” for claim 5 is both suspect and wrong.

For claim 9, the Petition is even more deficient. Claim 9 requires an “element attribute module.” For this element, the Petition points to the same tables as the “element module.” Petition, 44 (“Althoff’s relational class tables are the element attribute module.”) The Petition is identifying the same structure for the “element module” and the “element attribute module,” which is plainly wrong. These are two different modules. To the extent it appears that the Petition is pointing to a collection of multiple tables in Althoff, such a reading of the Petition would be mistaken because it would flatly contradict the (incorrect) claim construction on which the Petition is ostensibly based. Petition, 4.

D. Independent Claims 14 and 27

The Petition makes no unique or additional arguments for claims 14 and 27. Setting aside the arguments concerning “module,” a term absent from these claims, the rebuttal provided above, which focuses on claim 1, applies equally to these claims. Specifically, the analysis for each claim of Ground 1 is deficient at least because (1) the Petition attempts to combine aspects of Althoff’s relational database with Althoff’s object database even though the two databases are incompatible, and Althoff concerns a translator between these two databases specifically because they are incompatible, (2) Nye fails to fill the state machine gap found in Althoff because Nye discloses an event engine rather than a state machine, and thus lacks the state machine limitations, and (3) as discussed in the next sub-section, there is no motivation to combine Althoff and Nye.

E. No Motivation To Combine: Nye Teaches Away From The Proposed Combination

The Petition asserts that a skilled artisan would have been motivated to include Nye’s event model in Althoff database to “achieve the known benefits of combining a state machine with the OO model.” Petition, 10. Even assuming there were such “known benefits,” this argument fails because Nye does not disclose a “state machine,” but rather Nye discloses an “event model” that uses events as the nodes of a directed graph rather than states. Ex. 1005, 7:65-67.

Next, the Petition relies on Nye's statement that its event model "allows developers to add complex dependency logic to an existing database without having to modify the underlying structure of the database." Petition, 10. The critical language here is "without having to modify the underlying structure of the database."

Nye is thus suggesting the opposite of what Petitioner asserts would result from the combination of Althoff and Nye. The Petition asserts that Althoff's relational database is modified to include a state machine module and a status module, which would require the addition of one or more tables or columns of tables to Althoff's relational database. Petition, 12. Even assuming Nye disclosed a state machine and further disclosed these two modules, Nye teaches a "model" that is **separate from** the database it interacts with and thus can exist "without having to modify the underlying structure of the database." Ex. 1005, 5:33-36. Nye teaches the creation of a separate "event metamodel" that "is accomplished by maintaining values that represent the type of events" that may occur." *Id.*, 5:62-64. Nye explains: "Instead of changing the structure of the database a new data type can be created **in the event model**.... All of this is done without having to create new tables or change the relationship that exists between tables." *Id.*, 6:1-8. In sum, whereas the Petition asserts that a skilled artisan would add tables to Althoff's relational database, Nye actually states the opposite. A separate event

metamodel is used in conjunction with existing databases.

F. No Motivation To Combine: The Petition Fails to Explain How Althoff's Translation Process Would Work With Nye's Event Model

The purpose of Althoff's invention is to translate between an object-oriented database and a relational database: "[I]t would be advantageous to provide a method of compilation and translation between object-oriented and relational database structures which obtains the advantages of object-oriented database applications, while retaining the ability to use present relational database management systems." Ex. 1006, 3:4-9. Althoff's invention "allows a user to create, edit and manipulate a user's object database (which the method and system dynamically translates into a set of relational database structures), to create, edit and manipulate objects for that object database (which the method and system dynamically translates into data for those relational database structures)." *Id.*, 3:25-29.

Althoff translates the user's object database into a relational database by converting classes and their objects into a set of relational tables. Ex. 1006, FIGS. 1, 7, and 8. Althoff provides no teaching for the translation of event models designed for use with object-oriented databases into event models designed for use with relational databases. Nor does Nye provide any such teachings. The Petition quotes Nye as stating that its event model may be used on "relational databases,

object-oriented databases, or other database environments.” Petition, 12, Ex.1005, 6:39-43. However, Nye does not explain how one would translate an event model designed for one type of database to an event model for another type of database.

Accordingly, a skilled artisan would not modify Althoff based on the teachings of Nye. On the contrary, such a person would understand that Nye’s “event model” may be used with different types of databases, but would not know how to build a translator to translate the event model designed for the object database into an event model designed for the relational database. And without this knowledge, a skilled artisan would not be motivated to modify Althoff to add Nye’s event model, and certainly not with a reasonable expectation of success. The Petition fails to fill this critical gap.

The Petition treats Althoff as though it discloses a singular “database,” rather than a translator that translates between two different types of databases (a user’s object database in user model 230 and relational database 231). Ex. 1006, Fig. 2. Nye does not provide any teaching to apply its disclosure to such a system. The Petition asserts that a skilled artisan “would have readily understood that combining Althoff and Nye would enable event modeling in Althoff’s database, and thus further the functionality of Althoff’s system.” Petition, 12-13. However, there are at least two problems with this statement. First, Nye discloses that an event model works in conjunction with other databases, and does not disclose

modifying a database to include Nye’s functionality. In fact, the Petition confirms that Nye’s event model is used “without having to modify the underlying structure of the database.” Petition, 12. Second, Nye states that its event model works with relational database and object-oriented databases (although the details of how the event model interacts with either database are scant), but provides no teaching or suggestion how the event model would work with a system like Althoff with two databases and a translator. The Petition is deficient because it overlooks this critical incompatibility between Althoff and Nye.

VII. GROUND 2 BASED ON ALTHOFF IN VIEW OF NYE IN FURTHER VIEW OF SURJANTO IS DEFICIENT (CLAIMS 10, 23, 36)

This ground concerns dependent claims and fails to fill the gaps identified regarding the Althoff and Nye combination for Ground 1. Accordingly, Ground 2 is deficient for the same reasons as Ground 1.

VIII. GROUNDS 3 BASED ON ALTHOFF IN VIEW OF NYE IN FURTHER VIEW OF MOORE IS DEFICIENT (CLAIMS 6-7, 19-20, 32-33)

This ground concerns dependent claims and fails to fill the gaps identified regarding the Althoff and Nye combination for Ground 1. Accordingly, Ground 3 is deficient for the same reasons as Ground 1.

IX. SOTERA STIPULATION

In view of the Petitioner's representation that it is providing a "Sotera-type stipulation," Patent Owner will not address discretionary denial under *General Plastics*. Petition, 68. The denial of institution can and should be on the merits.

X. CONCLUSION

For the foregoing reasons, Patent Owner respectfully requests that the Board deny institution of Grounds 1-3 of the Petition, and thus decline to institute *inter partes* review of any claim of the '468 patent.

Respectfully submitted,

Date: October 18, 2023

/Steven R. Katz/

Steven R. Katz, Reg. No. 43,706
Andrew G. Pearson, Reg. No. 68,884
Jason W. Wolff, Reg. No. 43,281
Fish & Richardson P.C.
60 South Sixth Street
Minneapolis, MN 55402

Attorneys for Patent Owner

CERTIFICATION UNDER 37 C.F.R. §42.24(d)

Under the provisions of 37 C.F.R. §42.24(d), the undersigned hereby certifies that the word count for the foregoing Patent Owner's Preliminary Response to Petition totals 11,628 which is less than the 14,000 allowed under C.F.R. §42.24(b)(1).

Respectfully submitted,

Date: October 18, 2023

/Steven R. Katz/
Steven R. Katz, Reg. No. 43,706
Fish & Richardson P.C.
60 South Sixth Street
Minneapolis, MN 55402

Attorney for Patent Owner

CERTIFICATE OF SERVICE

Pursuant to 37 C.F.R. §§42.6(e), the undersigned certifies that on October 18, 2023, a complete and entire copy of this Patent Owner's Preliminary Response and associated exhibits were provided, via email to Petitioner by serving the correspondence email address of record as follows:

James M. Glass
John T. McKee
Quincy Lu
Joshua Parzivand
QUINN EMANUEL URQUHART & SULLIVAN, LLP
qe-servicenow-inquisient-ips@quinnemanuel.com
jimglass@quinnemanuel.com
johnmckee@quinnemanuel.com
quincylu@quinnemanuel.com
joshuaparzivand@quinnemanuel.com

/Crena Pacheco/
Crena Pacheco
Fish & Richardson P.C.
60 South Sixth Street, Suite 3200
Minneapolis, MN 55402
pacheco@fr.com