

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

SAP AMERICA, INC.,
Petitioner,

v.

ISIX IP LLC.,
Patent Owner.

IPR2024-00615
U.S. Patent No. 6,308,178

**DECLARATION OF DR. HENRY HOUH,
UNDER 37 C.F.R. § 1.68 IN SUPPORT OF PETITION FOR
INTER PARTES REVIEW**

TABLE OF CONTENTS

I.	Introduction.....	4
II.	Qualifications and Professional Experience	5
III.	Level of Ordinary Skill in the Art	9
IV.	Relevant Legal Standards	10
V.	Background.....	12
VI.	Overview of the '178 Patent.....	14
VII.	Claim Construction.....	19
VIII.	Identification of how the Claims are Unpatentable.....	20
A.	Overview of Prior Art.	20
1.	Summary of Brann.....	21
2.	Summary of Suresh.....	23
a.	Ground 1: Claims 1, 3, 4, 7 and 8 are obvious under 35 U.S.C. §103 over Brann.	25
1.	Claim 1.....	25
2.	Claim 3.....	45
3.	Claim 4.....	48
4.	Claim 7.....	50
5.	Claim 8.....	51
B.	Grounds 2 and 3: Claims 1 – 5, 7, 9, 10, and 23 – 25 are obvious under 35 U.S.C. §103 over Suresh alone or combined with Brann.	53
1.	Reasons to Combine Suresh with Brann	53
1.	Claim 1.....	55

2.	Claim 2.....	82
3.	Claim 3.....	83
4.	Claim 4.....	88
5.	Claim 5.....	92
6.	Claim 7.....	94
7.	Claim 9.....	96
8.	Claim 10.....	98
9.	Claim 23.....	99
10.	Claim 24.....	110
11.	Claim 25.....	110
C.	Grounds 4 and 5: Claims 8 and 19 are obvious under 35 U.S.C. §103(a) over Suresh in view Beauchamp or Suresh in view of Brann and Beauchamp	111
1.	Summary of Beauchamp.....	111
12.	Reasons to Combine Suresh and Beauchamp	112
13.	Claim 8.....	117
14.	Claim 19.....	119
IX.	Conclusion	125

I, Henry Houh, do hereby declare as follows:

I. INTRODUCTION

1. I am making this declaration at the request of SAP America, Inc. (“SAP”) in the matter of the *Inter Partes* Review of U.S. Patent No. 6,308,178 (“the ’178 Patent”) to Chang.

2. I have been asked to provide my opinions regarding whether claims 1-5, 7-10, 11-15, 19-20, and 23-25 (“the Challenged Claims”) of the ’178 Patent are unpatentable as they would have been obvious to a person having ordinary skill in the art (“POSITA”) at the time of the alleged invention, in light of the prior art. It is my opinion that all of the limitations of the challenged claims would have been obvious to a POSITA.

In the preparation of this declaration, I have studied:

- a. the ’178 Patent, Ex. 1001;
- b. U.S. Patent No. 5,842,205 to Brann, Ex. 1003;
- c. U.S. Patent No. 6,208,990 to Suresh et al., Ex. 1004;
- d. Prosecution History of U.S. Patent No. 6,308,178, Ex. 1005;
- e. U.S. Patent No. 6,621,505 to Beauchamp et al., Ex. 1006;
- f. Excerpts from Microsoft Press Computer Dictionary, Third Edition, Ex. 1009;
- g. U.S. Patent 6,014,670 to Zamanian et al., Ex. 1010;

- h. U.S. Patent No. 6,199,059 to Dahan et al., Ex. 1011;
 - i. U.S. Patent No. 6,738,975 to Yee et al., Ex. 1012; and
 - j. U.S. Patent No. 6,151,608 to Abrams et al., Ex. 1014.
3. In forming the opinions expressed below, I have considered:
- a. the documents listed above;
 - b. the relevant legal standards, including the standard for obviousness, and any additional authoritative documents as cited in the body of this declaration; and
 - c. my own knowledge and experience based upon my work in the field of wireless charging as described below.
4. Unless otherwise noted, all emphasis in any quoted material has been added.

II. QUALIFICATIONS AND PROFESSIONAL EXPERIENCE

5. My complete qualifications and professional experience are described in my *Curriculum Vitae*, a copy of which can be found in Exhibit 1017. The following is a brief summary of my relevant qualifications and professional experience.

6. I received a Ph.D. in Electrical Engineering and Computer Science from the Massachusetts Institute of Technology (“MIT”) in 1998. I also received a Master of Science degree in Electrical Engineering and Computer Science in 1991,

a Bachelor of Science degree in Electrical Engineering and Computer Science in 1989, and a Bachelor of Science Degree in Physics in 1990, all from MIT.

7. I am currently self-employed as an independent technical consultant. I am also president of a company that provides supplemental science, technology, engineering, and mathematics (“STEM”) education to children of all ages.

8. I worked as a summer intern at AT&T Bell Laboratories as part of a five-year dual degree program at MIT. I continued to work at AT&T Bell Laboratories as part of this MIT program. While I was at MIT, I was a teaching assistant (“TA”) in the Electrical Engineering and Computer Science Department’s core Computer Architectures course. I first was a TA as a senior for a role typically reserved for graduate students. I later became head TA. The course covered various topics in computer architectures, including memory caching, different types of cache, and cache replacement strategies. As a TA, I helped write homework assignments, lab assignments, and exams. I also taught in the recitation sections.

9. Later, as part of my doctoral research at MIT from 1991-1998, I was a research assistant in the Telemedia Network Systems (“TNS”) group at the Laboratory for Computer Science. The TNS group built a high-speed gigabit network and created applications that ran over the network. Example applications included ones for remote video capture, processing, and display of video on

computer terminals. In addition to working on the design of core network components, designing and building the high speed links, and designing and writing the device drivers for the interface cards, I also set up the group's web server.

10. The TNS group I researched and installed web servers and web caches. I also helped to build the web pages that initiated these video sessions via a web interface. Vice-President Al Gore visited our group in 1996 and received a demonstration of – and remotely drove – a radio controlled toy car with a wireless video camera mounted on it; the video was encoded by TNS-designed hardware, streamed over the TNS-designed network and displayed using TNS-designed software.

11. I co-authored and presented a paper with a colleague about our video demonstrations on the web at the first World Wide Web conference held in Geneva, Switzerland at CERN. CERN is the birthplace of the Web, and the conference was hosted by Tim Berners-Lee, the inventor of the Web. In addition, we also presented a paper that provided dynamic web pages based on data extracted from a database, which we called an “Active Page.” We allowed users to edit and correct data in the database to demonstrate full integration of a web page with a database.

12. From 1997 to 1999, I was a Senior Scientist and Engineer at NBX

Corporation, a start-up that made business telephone systems for streaming packetized audio over data networks instead of using traditional telephone lines.

13. From 1999-2004, I was employed by Empirix or its predecessor company, Teradyne. Empirix was a leader in test tools for telecommunications protocols and systems, providing functional testing tools as well as load testing tools. At Empirix I worked with the web test division to re-architect the division's primary product for functional and load testing of web sites. The tool could pull data from a database to provide variation of data submitted to the web applications, and then to extract data from a web page to populate a database with key results computed by the web site after customized data is submitted.

14. I have designed numerous web-based applications that utilized databases, including designing the schemas. I debugged and reconstructed live databases when data was corrupted, have deployed open source web applications that integrate with databases, and have written custom SQL scripts to extract data from databases.

15. I have been awarded numerous United States patents, and I have several patent applications pending. The patents and patent application are listed in my *Curriculum Vitae*.

16. I am being compensated for my work associated with this case plus reimbursement of reasonable expenses. My compensation is not contingent on my

opinions or the outcome of the case, and I have no other interest in this case or the parties thereto.

III. LEVEL OF ORDINARY SKILL IN THE ART

17. I understand there are multiple factors relevant to determining the level of ordinary skill in the pertinent art, including (1) the levels of education and experience of persons working in the field at the time of the invention; (2) the sophistication of the technology; (3) the types of problems encountered in the field; and (4) the prior art solutions to those problems.

18. I understand there are multiple factors relevant to determining the level of ordinary skill in the pertinent art, including (1) the levels of education and experience of persons working in the field at the time of the invention; (2) the sophistication of the technology; (3) the types of problems encountered in the field; and (4) the prior art solutions to those problems.

19. A person of ordinary skill in the art (“POSITA”) in the field of the ’178 Patent, as of its earliest possible filing date of October 21, 1999, would have been someone knowledgeable and familiar with integration systems that are pertinent to the ’178 Patent. That person would have a bachelor’s degree in computer science, computer engineering, or equivalent training, and approximately two years of experience working in the computer science or engineering field. Lack of work experience can be remedied by additional education, and vice versa.

20. For purposes of this Declaration, in general, and unless otherwise noted, my statements and opinions, such as those regarding my experience and the understanding of a POSITA generally (and specifically related to the references I consulted herein), reflect the knowledge that existed in the field as of the earliest possible priority date of the '178 Patent (i.e., Oct. 21, 1999). Unless otherwise stated, when I provide my understanding and analysis below, it is consistent with the level of a POSITA prior to the alleged priority date of the '178 Patent.

IV. RELEVANT LEGAL STANDARDS

21. I am not an attorney. In preparing and expressing my opinions and considering the subject matter of the '178 Patent, I am relying on certain basic legal principles that counsel have explained to me. These principles are discussed below.

22. I understand that prior art to the '178 Patent includes patents and printed publications in the relevant art that predate the priority date of the alleged invention recited in the '178 Patent. For purposes of this Declaration, I am applying October 21, 1999 date as the earliest possible priority date of the '178 Patent.

23. I have been informed that a claimed invention is unpatentable under 35 U.S.C. § 103 if the differences between the invention and the prior art are such that the subject matter as a whole would have been obvious at the time the

invention was made to a person having ordinary skill in the art to which the subject matter pertains. I have also been informed by counsel that the obviousness analysis takes into account factual inquiries including the level of ordinary skill in the art, the scope and content of the prior art, and the differences between the prior art and the claimed subject matter.

24. I have been informed by counsel that the Supreme Court has recognized several rationales for combining references or modifying a reference to show obviousness of claimed subject matter. Some of these rationales include the following: (a) combining prior art elements according to known methods to yield predictable results; (b) simple substitution of one known element for another to obtain predictable results; (c) use of a known technique to improve a similar device (method, or product) in the same way; (d) applying a known technique to a known device (method, or product) ready for improvement to yield predictable results; (e) choosing from a finite number of identified, predictable solutions, with a reasonable expectation of success; and (f) some teaching, suggestion, or motivation in the prior art that would have led one of ordinary skill to modify the prior art reference or to combine prior art reference teachings to arrive at the claimed invention.

25. I further understand that certain factors may support or rebut the obviousness of a claim. I understand that such secondary considerations include,

among other things, commercial success of the patented invention, skepticism of those having ordinary skill in the art at the time of invention, unexpected results of the invention, any long-felt but unsolved need in the art that was satisfied by the alleged invention, the failure of others to make the alleged invention, praise of the alleged invention by those having ordinary skill in the art, and copying of the alleged invention by others in the field. I understand that there must be a nexus—a connection—between any such secondary considerations and the alleged invention. I also understand that contemporaneous and independent invention by others is a secondary consideration tending to show obviousness.

26. I am not aware of any allegations by the named inventor of the '178 Patent or any assignee of the '178 Patent that any secondary considerations are relevant to the obviousness analysis of any Challenged Claim of the '178 Patent.

V. BACKGROUND

27. Integrating data among different software products was a well-known problem. This problem was exacerbated by the dot com boom in the 1990s that saw a plethora of new technology companies building and commercializing their own software products and selling these products to a variety of businesses. A business purchasing these software products for its business needs would often end up having numerous, disparate software applications, each with its own data model. As the background section of the '178 Patent explains, these data models

can be relational models, spreadsheets, multimedia databases, knowledge databases, and other types of data models. '178 Patent, 1:25-30. As a result, a business using different software products would often dedicate significant human and capital resources to integrating data among the different software products. This is because data integration was a necessary evil that allowed a business to readily access, compile, and analyze data for its business needs. Suresh, 1:24-27.

28. A common solution among businesses was to develop custom integration software that solved each business's unique data integration needs. This solution, however, was fraught with drawbacks, including 1) a never ending development cycle that incorporated changes and updates to the existing data models and added new data models from new software products to the integration system, 2) time consuming software code and bug fixes, and 3) a need for experts well-versed in software programming languages, such as SQL, C++, Cobol, etc., who understood the intricacies of different data models. Suresh, 2:22-47; '178 Patent, 1:55-2:1. Accordingly, businesses began to look to other integration solutions that reduced these drawbacks.

29. The purported invention in the '178 Patent describes an end-to-end integration solution that integrates data among heterogeneous source and destination applications, including data in heterogeneous databases. '178 Patent, 1:32-41, 3:53-65, Figure 1. This integration solution was already known in the art.

An integration system that used a control program that activated subroutines in a specified order to transform data from a source to a target in the same way as the claimed invention of the '178 Patent was known prior to the filing of the '178 Patent. Brann, Abstract, 7:27-30; 7:55-57.

30. Further, this solution was already known in the database art. Specifically database integration systems that used an ETL (Extract/Transport/Load) architecture to integrate data from a variety of operational databases into a variety of data warehouses and data marts already existed and performed the purported invention in the '178 Patent long before the '178 Patent was filed. Suresh, 2:59-67, Figure 2.

31. Accordingly, as will be described in more detail below, the '178 Patent describes and claims no more than what was already known in the art with regard to integrating data among different applications.

VI. OVERVIEW OF THE '178 PATENT

32. The '178 patent generally relates to an integration system “for providing migration of data among heterogeneous applications.” '178 Patent, 1:5-10. As explained in the background, businesses often need “to integrate various business software applications employed by the diverse operating sectors of ... [the] organization (for example, financial operations, human resources, shipping and receiving, etc.).” '178 Patent, 1:14-25. Because the various sectors of a

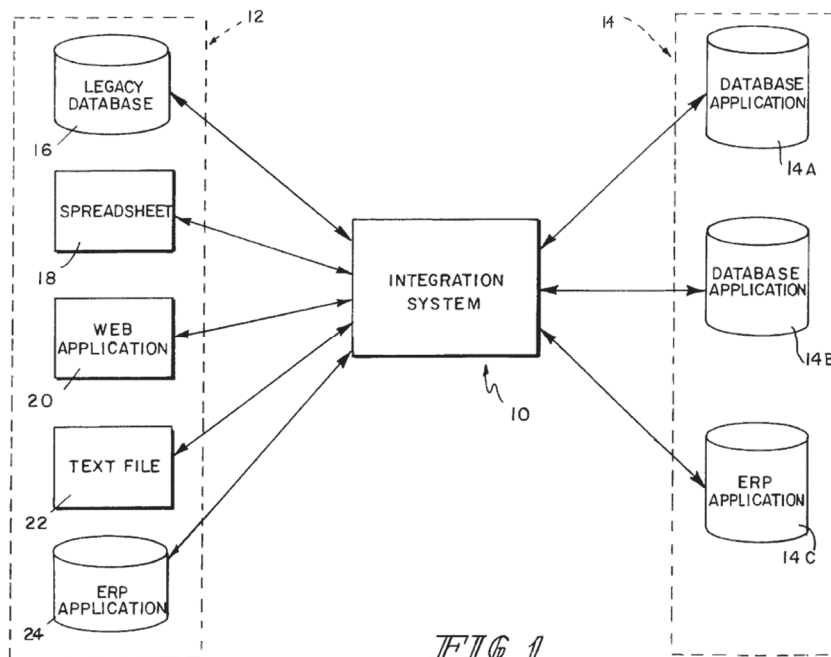
company may use different software applications, it has long been known to use systems for integrating the various software applications so that they interface properly. The '178 patent claims such a system for integrating data among heterogeneous source applications and destination applications. '178 Patent, 9:51-53.

33. Independent claim 1 recites just two main elements—that a repository and an active component. The “repository includes of information” is simply a store of information about the source and destination applications, and includes data elements that perform operations on data. '178 Patent, 9:54-60. In particular, the repository includes “data elements” and “a first module.” The data elements are analogous to subroutines. *See Id.* at 6:33-37. A first module in the repository stores instructions sets for activating the data elements, that is, these instruction sets define which subroutines are called in what order. *Id.* These subroutines provide different functions so that, rather than creating a custom program for integration, the repository 40 simply includes a library 50 of subroutines that may be combined as needed. A first module 70 in the repository 40 stores instructions sets for activating the data elements 60, that is, these instruction sets define which subroutines are called in what order. '178 Patent, 6:54 -60, 9:54-60 . *Id.* The “first module” is a pre-built integration package for a specific application. '178 Patent, 2:47-51. The first module includes rules for how the subroutines are combined so

that the program just picks from the library which subroutines to use for data integration with the particular destination application, instead of providing a custom program that includes only the necessary subroutines. *Id.* at 2:53-56, 3:5-16.

34. Regarding the “active component,” that element includes an “input” and an “output.” ’178 Patent, 9:61-64. The input simply receives the data from the source application 12. *Id.* The output simply populates the destination application 14 with that data processed in accordance with by the data elements 60 called from the repository 40.

35. The integration system is depicted in Figure 1, below:



’178 Patent, Figure 1.

36. As illustrated in Figure 1, a source applications 12 is simply the

application that has the data which needs to be integrated with another application, which is the destination application 14. The source applications 12 “include any combination of legacy database applications 16, spreadsheet applications 18, web applications 20, applications containing text files 22, or even ERP applications 24” and the destination applications 14 “include any combination of database applications 14A, 14B and ERP applications 14C.” ’178 Patent, 4:7-12.

37. The integration system includes an “active component 30” and a “repository 40.” ’178 Patent, 4:17-20. This is depicted in Figure 5, below:

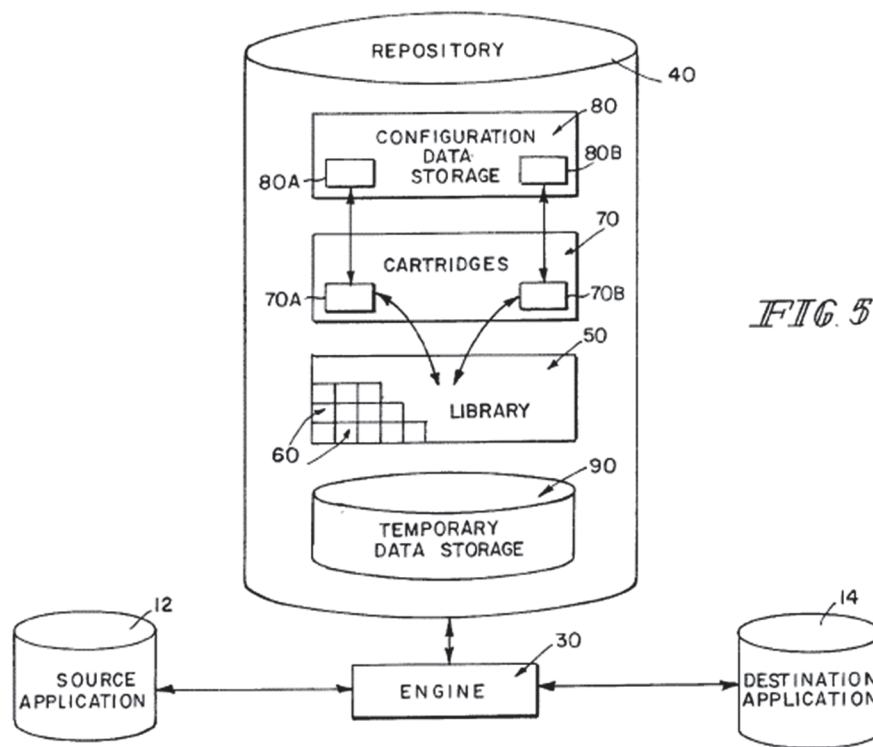
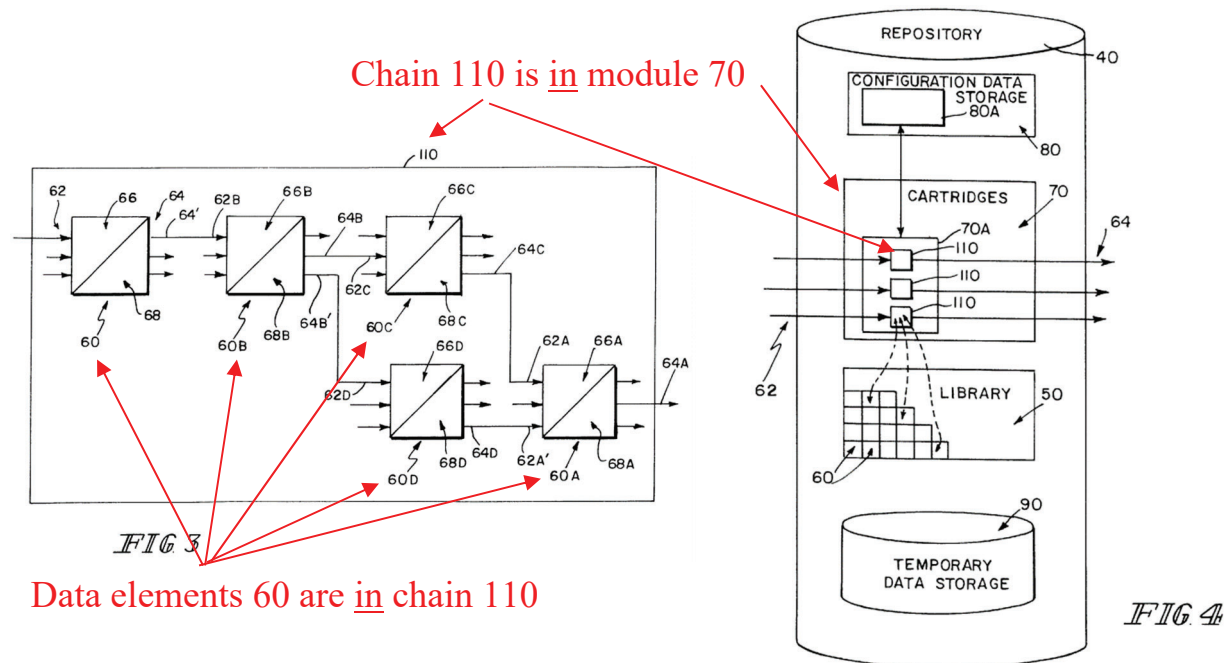


FIG. 5

’178 Patent, Figure 5.

38. The repository 40 “stores business logic that may relate to the entire suite of [destination] applications 14.” ’178 Patent, 4:42-44. Repository 40

includes “data elements 60” and “plug-in modules 70”. ’178 Patent, 4:20-33. The “data elements 60 essentially consist of reusable code or units of work that perform a discrete data manipulation function, and may be shared among interfaces defined by modules 70.” ’178 Patent, 6:33-36. Modules 70 are described as cartridges that provide “software add-ons configured to operate with active component 30.” ’178 Patent, 4:20-33, 2:44-47. Modules 70 “assemble the instructions by accessing data elements (individual units of re-usable code) stored in a repository.” ’178 Patent, 3:5-8. The specification describes that the data elements are “assembled into combinations of data elements (data element chains 110),” which may be inside the module. ’178 Patent, 6:36-38.



’178 Patent, Figure 3 and 4 (annotated)

39. The active component (shown as engine 30 in Figure 5 above),

“performs the underlying communication and data processing functions.” ’178 Patent, 2:40-44. The active component uses “instruction sets associated with the applicable module and other information in the repository to read and process inputs from a source application and perform the transfer of data to a destination application.” ’178 Patent, 2:56-60. For example, when “active component 30 provides an input 62 to module 70A,” module 70A provides an instruction set which “calls upon a series of data elements 60” to perform individual tasks “to achieve desired output 64.” ’178 Patent, 6:48-60.

40. As I explain below, the concept of an integration system having these common components was not new as of the earliest alleged priority date of the ’178 patent.

VII. CLAIM CONSTRUCTION

41. It is my understanding that in order to properly evaluate the ’178 Patent, the terms of the claims must first be interpreted. It is my understanding that for the purposes of this *inter partes* review, the claims are to be construed under the so-called *Phillips* standard, under which claim terms are given their ordinary and customary meaning as would be understood by one of ordinary skill in the art in light of the specification and prosecution history, unless the inventor has set forth a special meaning for a term. For the purposes of this declaration, no terms require specific construction, other than the following:

The preamble is limiting. (Claim Construction Order (ECF No. 37), No. 6:22-cv-201-ADA-DTG, Isix IP LLC v. Int'l Bus. Machines Corp., Jan. 4, 2023.).

“data element(s)” was construed as “individual unit(s) of reusable code.” *Id.*

VIII. IDENTIFICATION OF HOW THE CLAIMS ARE UNPATENTABLE

42. I have been asked to provide my opinion as to whether the Challenged Claims of the '178 Patent would have been obvious in view of the prior art. The discussion below provides a detailed analysis of how the prior art reference identified below teaches the limitations of the Challenged Claims of the '178 Patent.

43. As part of my analysis, I have considered the scope and content of the prior art and any differences between the alleged invention and the prior art. I describe in detail below the scope and content of the prior art, as well as any differences between the alleged invention and the prior art, on an element-by-element basis for each Challenged Claims of the '178 Patent.

44. As described in detail below, the alleged invention of the Challenged Claims would have been obvious in view of the teachings of the identified prior art references as well as the knowledge of a POSITA.

A. Overview of Prior Art.

45. As described in detail below, the alleged invention of the Challenged Claims 1-5, 7-10, 19 and 23-25 would have been obvious in view of the teachings

of the identified prior art references as well as the knowledge of a POSITA.

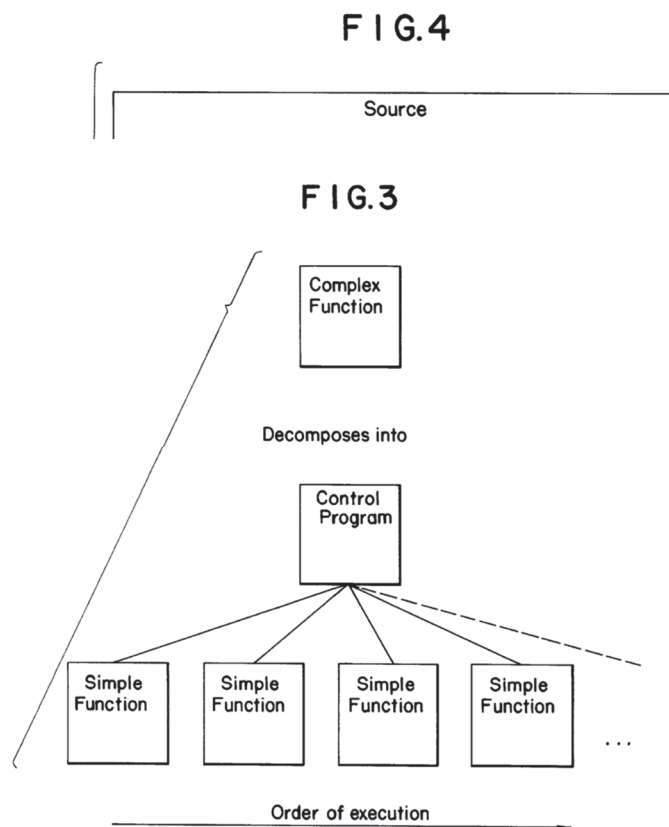
1. Summary of Brann

46. Brann, assigned to a subsidiary of Petitioner SAP America, Inc., teaches a system for integrating data among source and destination applications. Brann, 1:10-13, 5:27-28. A control program activates subroutines, stored in a database, in a specified order to transform data from the source to the target. Brann, Abstract, 7:27-30; 7:55-57. The control program accesses a database to obtain its instructions. *Id.* The database stores information about the source and target, as well as all information needed for the transformation. Brann, Abstract, 3:49-50.

47. Brann describes a mechanism for “the rapid integration of computer systems.” 5:27-28. Brann provides a “process for providing an efficient system for assembling and for translating information from one computer to another.” Brann, 1:10-13. Just as in the ’178 patent, Brann recognized and solved the problem that “[o]ver time, the number of independent systems has grown enormously. There has been little attention paid to coordination of the data between the competing programs.” Brann, 1:16-30. Brann also recognized, like the ’178 patent, that using a new computer program as a go-between for heterogeneous applications requires custom, manual programming. Brann, 2:59-62. (*See* ’178 patent at 1:55 – 2:23).

48. The system of Brann solved the problem of integrating data between a

source application (using a source message from a computer on one type of computer system) and a destination application (using a target message for a computer program on a second type of computer) prior to and in the same manner as the '178 patent. Figure 4 of Brann reproduced below illustrates Brann's integration of data among source and target applications.



Brann, fig. 4 (annotated)

49. As shown in Figure 3, Brann utilizes a control program that calls subroutines to perform simple functions to carry out the transformation of data between source and target.

50. Brann states that functions for transformations are “are best implemented as subroutines which can be re-used in similar instruction needs.” Brann, 6:29-36. The “subroutines are capable of being assembled into any conceivable Translation Program.” Brann, 9:13-14. Brann also describes a repository in the form of a database. The database stores the “data and instructions corresponding to the steps necessary to translate the source message to the target message.” Brann, 4:6-8.

2. Summary of Suresh

51. Like the '178 patent, Suresh is directed to data integration through a “data extraction/transformation/loading (ETL) process.” Suresh, 4:55-57. The system in Suresh is depicted in Figure 2, below:

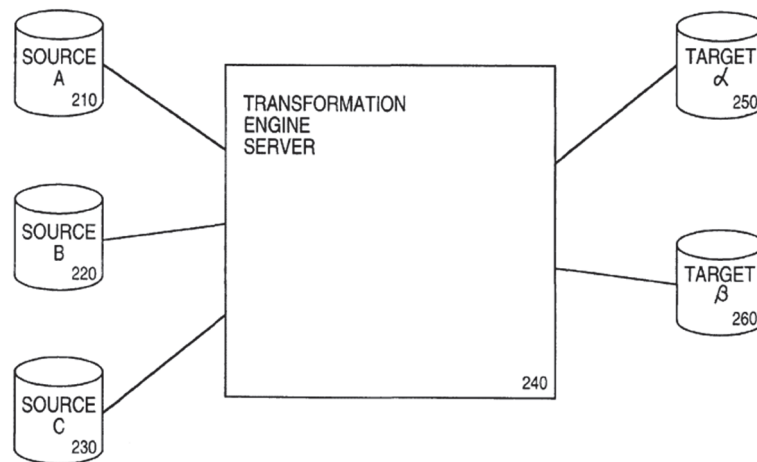


Figure 2

Suresh, Figure 2

52. The Source A, Source B, and Source C in Suresh are “operational databases 210, 220, and 230 [that] store the data resulting from business and financial transaction, and/or from equipment performance logs.” Suresh, 6:8-11. The operational databases 210, 220, and 230 “provide the data source from which the transformation engine server 240 can extract, transform, then load the data into the databases 250 and 260.” Suresh, 6:15-20. The “[d]atabases 250 and 260” are “warehouses or data marts” and are shown as Target α and Target β in Figure 2. Suresh, 6:19-20.

53. The transformation engine server 240 in Suresh performs the ETL process using a “cascading sequence of transformation components” called a pipeline. Suresh, 7:9-12. As illustrated in Figures 4-8 and 9 of Suresh, there may be different types of pipelines that include different cascading sequences of transformation components. Suresh, Figures 4-8, 9. Like the data elements in the ’178 Patent, the transformation components are “ready-made” pieces of source code where each transformation component “is responsible for one specific type of transformation.” Suresh, 6:46-52; 6:61-65.

54. The computer system in Suresh also includes a repository. Suresh, 6:23-29. Suresh describes its repository as keeping “track of session information” plus storing all the “mapping information relating to how data is to be mapped and transformed from target tables of the operational databases to target tables of data

ments.” Suresh, 6:23-29.

55. Suresh therefore illustrates that integration systems that transform data from source applications to destination applications were already well known in the art prior to the earliest alleged priority date of the '178 Patent.

a. Ground 1: Claims 1, 3, 4, 7 and 8 are obvious under 35 U.S.C. §103 over Brann.

1. Claim 1

[1.0] *A system for integrating data among heterogeneous source applications and destination applications, comprising:*

56. To the extent that the preamble is limiting, Brann describes and suggests the claimed “system for integrating data” as it describes a mechanism for “the rapid integration of computer systems.” Brann, 5:27-28. Brann provides a “process for providing an efficient system for assembling and for translating information from one computer to another.” Brann, 1:10-13. The system includes “a control program” and “a database.” Brann, 21:47-58. The control program executing on a computer accesses the database to look up and obtain information, thus forming a system. *Id.*; (“providing a computer, having a control program executing therein.”) Brann, 14:33-34; (“providing a database, associated with the computer.”) Brann, 14:41-42.

57. The system of Brann integrates “*data among heterogeneous source*

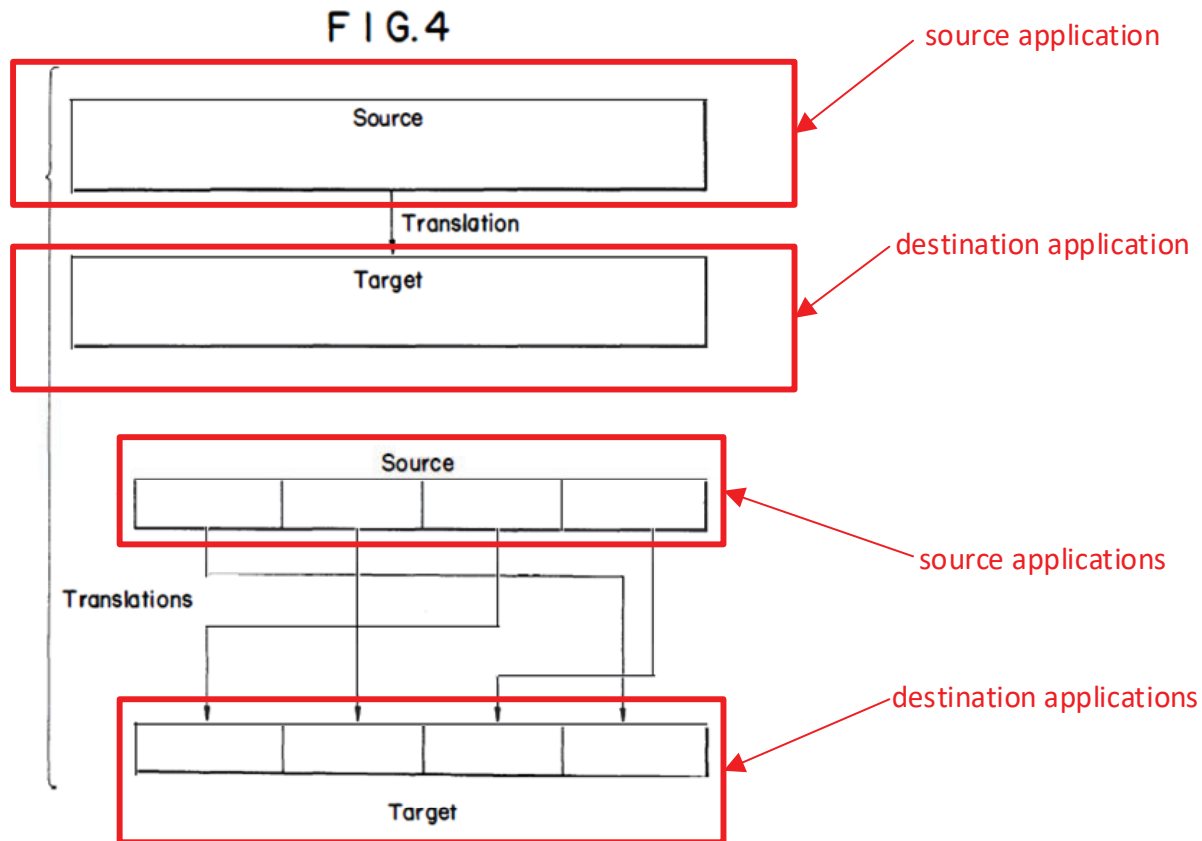
applications and destination applications.” Brann recognized and solved the problem that “[o]ver time, the number of independent systems has grown enormously. There has been little attention paid to coordination of the data between the competing programs.” Brann, 1:16-30. Brann recognizes that different types of competing computer programs (applications) utilize and process data differently, “another difficulty lies in the interaction between different systems.” Brann, 2:54-55. Brann also states that it is important “that the data contained in these systems can be used in later-developed systems,” thus recognizing the importance of integrating data. Brann, 2:49-53.

58. A POSITA would recognize that the “competing programs” described in Brann includes “applications,” as “program” and “application” were used interchangeably at the relevant time. For example, an “application” is a computer program designed to assist in the performance of a specific task. Microsoft Computer Dictionary, pg. 27.

59. In addition, the programs of Brann are executed on “different” and “independent” computer systems. Brann thus describes that the computer systems and programs are of “different types,” that is they are heterogeneous. A POSITA would recognize that since the computer systems are heterogeneous, the applications running on those heterogeneous computer systems are also heterogeneous. Brann, 2:45-55. For example, a POSITA at the relevant time

understood that data is transformed when an upgrade is performed or moved to a different, heterogeneous platform. Brann teaches “that different computer systems represent information in different forms. It is well known that even in systems provided by the same manufacturer, the data structures describing objects in the real world are different in different systems, even when the real-world objects being described are identical.” Brann, 7:9-15.

60. Moreover, Brann describes and suggests that data sent from “source applications” to “target applications” is translated or transformed. Brann, 11:7-30. There would be no need to translate and/or transform the data if the source and target were the same. A POSITA would recognize that the source (source application) and target (destination application) are heterogeneous. Annotated Figure 4 of Brann below illustrates Brann integrating data among source applications and destination applications via translation(s).



Brann, fig. 4 (annotated)

61. See also 7:29-52; 11:46-47; 14:33-34 of Brann.

62. As such, a POSITA would understand Brann also describes that these heterogeneous applications are heterogeneous source applications (source) and destination applications (target).

[1.1] a repository of information relating to the source applications and the destination applications, the repository including a plurality of data elements each being capable of performing a discrete operation on a piece of data, and

63. Just like the '178 patent, Brann describes and suggests a database as

“a repository of information.” Brann, 4:4-8; 12-15 and 19-25. The ’178 patent states “[c]onfiguration data of configuration data storage 80, data elements 60 of library 50, and temporary data of temporary data storage 90 are all stored as Oracle tables within a relational **database represented by repository 40.**” (emphasis added). ’178 patent, 4:33-37. As such, the ’178 patent states that the repository may be a database. A POSITA at the relevant time would recognize this as an option.

64. In a similar manner, the database in Brann stores information relating to the source applications and the destination applications. For example, Brann discloses “storing instructions, data, and the structures of any source stimuli to be acted upon in the database.” Brann, 3:49-51. A POSITA would understand this to be information relating to the source and destination applications.

65. In addition in Brann, information describing the message types used by the source and target, which is information relating to the source and destination applications, is stored in tables in the database.

“Each of the incoming message types is recorded in the source table together with any arbitrary data to be found in messages of this type in the section defined in the location table. Each of the target messages for each source message is recorded in the target table. Each of the details of the transformation is recorded in a translation details table.” Brann, 11:4-10.

66. The database also stores “data and instructions corresponding to the steps necessary to translate the source message to the target message.” Brann, 4:6-8. At least the above types of information stored in the database of Brann are “*information relating to the source applications and the destination applications.*”

67. The database of Brann also “include[s] the plurality of data elements *each being capable of performing a discrete operation on a piece of data.*”

68. The claim term “data element(s)” was construed as “individual unit(s) of reusable code.” (Claim Construction Order (ECF No. 37), No. 6:22-cv-201-ADA-DTG, Isix IP LLC v. Int’l Bus. Machines Corp., Jan. 4, 2023.).

69. Just like the ’178 patent, Brann describes and suggests reusable subroutines (data elements) that perform discrete operations. For example, the ’178 patent states that the data elements “essentially consist of reusable code or units of work that perform a discrete data manipulation function, and may be shared...” ’178 patent. 6:33-35. Just as in the ’178 patent, Brann discloses “an atomic unit of processing appropriate to the task in hand. In fact, in the construction of computer systems, the form of the ‘instructions’ is usually defined by the task at hand and is then expressed using a suitable programming language. It is well known that such functions are best implemented as subroutines which can be re-used in similar instruction needs.” Brann, 6:29-36. Brann thus describes and suggests the claimed “*data elements.*”

70. The subroutines of Brann are “*each being capable of performing a discrete operation on a piece of data.*” Brann states that a subroutine “carries out the particular transformation required.” Brann, 11:11-12. Brann states that respective subroutines may be used “[t]o carry out functions like:

Copy (with justification to left or right, as appropriate)

Convert from character to integer numeric formats (and vice versa)

Convert from character to floating point numeric formats (and vice versa)

Convert from floating point to integer numeric formats (and vice versa)

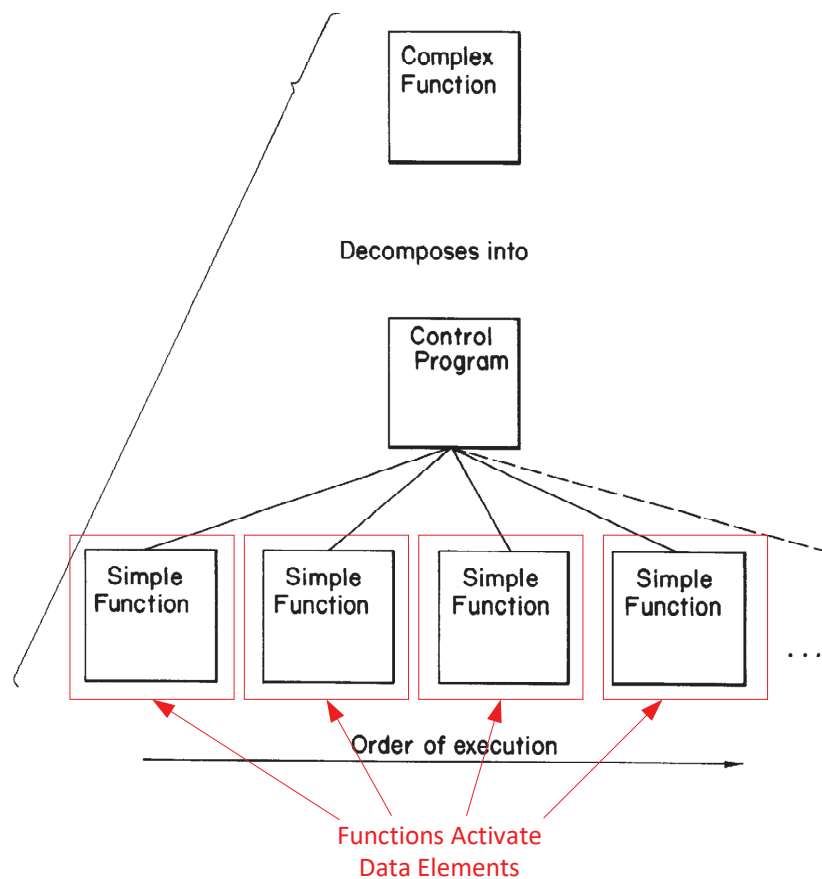
Lookup incoming data file in table of values (output result as characters, integer, or floating point)

A further list of representative functions is attached in the appendix. When invoked, each of these routines is passed a reference to the data it must take as input, a reference to the place its output must be written to, the maximum allowable lengths of input and output, and a reference to the lookup table (if any) being employed.” Brann, 8:38-55; *See also* Brann, 16:10-17:37.

71. Brann also describes the subroutines as “methods,” and describes various types of methods, such as Translation Methods, identification Methods, and Application Methods. Brann, 9:10-13. “These subroutines are capable of being assembled into any conceivable Translation Program.” Brann, 9:13-14.

72. “Translation Methods are written as subroutines which receive a

particular set of parameters. Each Method carries out a particular type of transformation. For example, there is one to do copying and one to do conversion of character format numbers into integers. Each of the Methods carries out a particular type of manipulation of simple data types.” Brann, 12:47-52.

FIG. 3

Brann, Fig. 3 (annotated).

73. Annotated Figure 3 of Brann above shows the simple functions (subroutines) invoked by the control program to perform an operation. Brann thus

describes and suggests that the subroutines are used to perform “*discrete operations on pieces of data*” as claimed.

74. Moreover, a POSITA would understand the “task in hand” and “particular transformation” performed by the subroutines in Brann are types of “*discrete operations*.” A subroutine is a discrete operation since it is so labeled, regardless of what the subroutine is comprised of (e.g., multiple other operations). Brann renders obvious “*data elements*” (subroutines) that perform a “*discrete operation on a piece of data*” (task in hand, etc.).

75. Brann’s translation method programs, also called subroutines, correspond to the data elements of the ’178 patent. (“Writing a translation method program for each transformation of each type of element of the source message to each type of element of the target message.”). Brann, 4:9-11

76. Brann also describes and suggests that the subroutines are included in the database. As described above, Brann describes and suggests that the subroutines are used to carry out the transformations. Brann, 7:22-30. Brann states that the subroutines are stored: “It is still a further object of this invention to provide a method of creating programs by storing and calling upon basic programming elements [subroutines] and combining those elements for use at the program's execution-time.” Brann, 3 :5-8. Brann also states that the steps for the transformation are described in the “database in which all necessary instructions

and data are stored.” Brann, Abstract. Brann describes throughout that all of the instructions and data needed to perform a translation are stored in the database. *See* Brann, Abstract, 3:5-8, 3:19; 4:6-7, 4:53-55, 14:63-64, 20:64-67. As such, a POSITA would understand that Brann describes that the subroutines are stored in the database.

77. To the extent that the Brann does not explicitly describe that the subroutines are stored in the database, it would have been obvious to a POSITA at the time of invention to do so. A POSITA would have recognized that the database of Brann would also be used to store the subroutines used for the translation, along with storing the translation instructions, data, and the like. A POSITA would have done so to maintain all of the information together. Brann teaches that “late binding programs described consist of two parts – a Control Program and subroutines, which carry out the required program steps.” Brann, 7:21-23. For example, Brann further teaches that late binding programs comprise “storing instructions and data in the database.” Brann, 3:14-19. Thus, a POSITA would have found it obvious to store subroutines, as part of the late binding programs, in the database. There would have been a reasonable expectation of success to do so as Brann already describes storing information, such as instructions and data in the database.

78. Thus, Brann renders this claim term obvious.

[1.2] *[the repository including...] a first module including a plurality of instruction sets, each instruction of each instruction set activating a data element; and*

79. The database (*repository*) of Brann includes a plurality of transformation instructions (*instructions sets*) for the control program to perform the transformation. The transformation instructions are stored in the database, for example in rows of a database table, which a POSITA would understand corresponds to the “*first module*,” As the “*first module*” may be a collection of instructions sets. ’178 Patent, 2:53-57, 8:55-56.

80. Brann states that the “Control Program extracts its instructions from the database.” Brann, 7:33-34. The instructions of Brann include a “list of subroutines to be invoked and an order in which they should be invoked. The Control Program then invokes the required subroutines in the required order.” Brann, 7:26- 27. This stored “list of subroutines to be invoked and an order in which they should be invoked” describes and suggests the “*first module including a plurality of instruction sets*,” and is described in Brann as a set of “transformation instructions,” Brann, 10:58-59, and also as a “Translation Program.”

81. Brann further discloses “storing instructions in the database identifying which translation method programs are to be called.” Brann, 4:15-16.

(Also “storing in the database data and instructions corresponding to the steps necessary to translate the source message to the target message.” Brann, 4:53-56).

The subroutines are assembled into the Translation Program to carry out the transformation. Brann, 9:13-14. “The set of subroutines required to implement a Translation Program consists, therefore, of a set of structural conversion routines.” Brann, 8:36-38. Brann also discloses “causing the control program to retrieve information from the database relating to a temporal sequence of execution of a plurality of response generation programs, said causing the control program to respond step further comprising in the temporal sequence, invoking each of the plurality of response generation programs for execution on the computer.” Brann, 18:30-36.

82. Brann goes on to explain that the set of transformation instructions include “the originating data elements (i.e., a field in the first structure), a resulting data element (i.e., a field in the second structure), and the Transformation Method required between them.” Brann, 10:59-63. Brann describes that the database includes a plurality of database tables, the database tables having rows for storing the different transformation instructions. The transformation instructions are entered into a row (first module) in a table in the database. Brann, 10:58-26 “Each of the details of the transformation is recorded in a translation details table.” The database tables are populated with the above information, and the Control Program

interprets this information “as instructions for carrying out the translation.” Brann, 11:26-30. “When the Control Program has executed all the Methods [subroutines] it is directed to do, it delivers the outgoing structure to the receiver. Brann, 12:6-8.

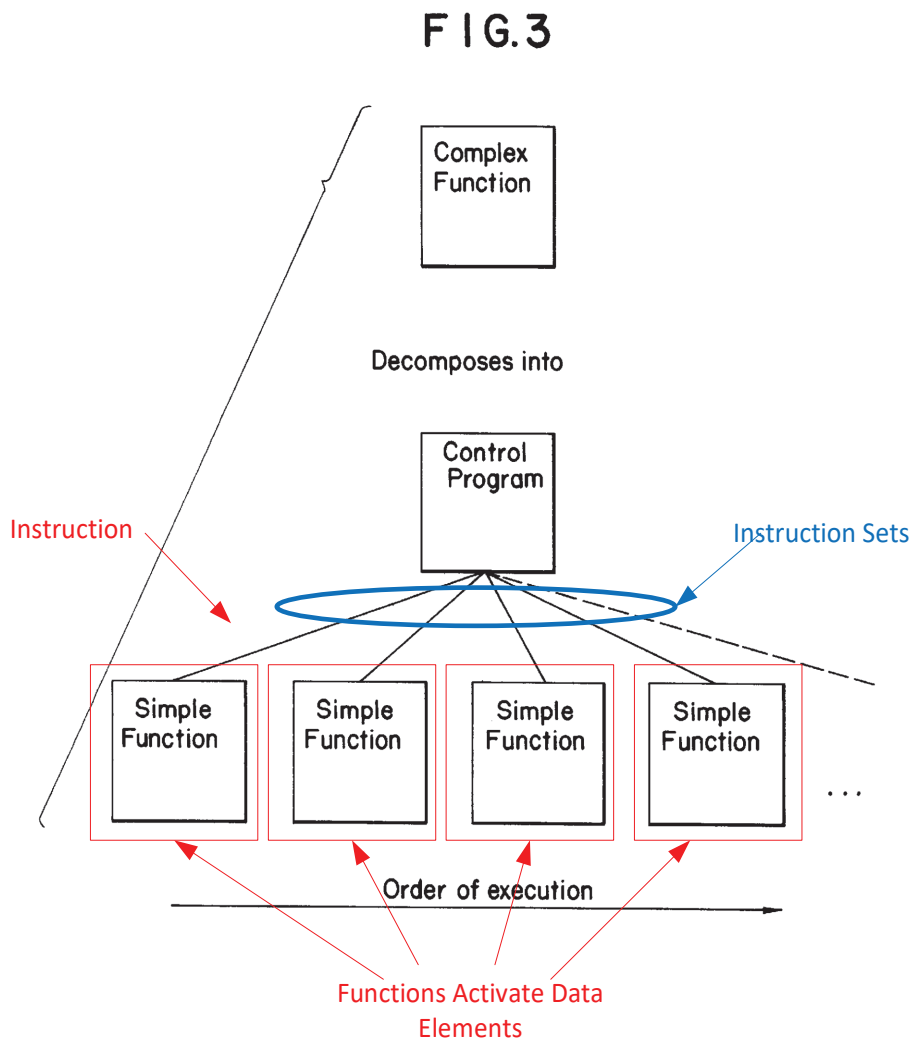
83. A POSITA would understand the database storing the “temporal sequence of execution” of the subroutines, Brann, 18:30-36, for example as database tables having rows storing the set of transformation instructions for the various translations, as a “*first module including a plurality of instruction sets.*”

84. Brann also describes and suggests “*each instruction of each instruction set activating a data element*” as each of the transformation instructions activating a subroutine by instructing the Control Program to call that subroutine. The ’178 patent describes this claimed function as “[m]odule 70A provides an instruction set for achieving the output which then calls upon a series of data elements 60 within library 50 to perform individual tasks that may be common to a variety of modules 70.” ’178 patent, 6:54-58. “[M]odules 70 do not actually contain data elements 60, but rather contain information describing where they may be found in library 50.” ’178 patent, 7:38-41.

85. In a similar manner, Brann discloses “the database contains a unique string of characters for each available subroutine.” Brann, 7:45-46. “That name is used by the Control Program to derive the executable function reference of the appropriate subroutine when it reads the database.” Brann, 11:13-17. The control

program “resolves the reference to find the subroutine and execute [sic] the subroutine.” Brann, 11:19-20. Thus, the database and database tables in Brann contain information describing where the subroutines may be found, just as the “module” described in the ’178 patent.

86. Annotated Fig. 3 of Brann below illustrates how the instructions in the instruction set are used to activate a data element.



Brann, Fig. 3 (annotated)

87. The lines between the control program and each simple function represents that the control program is composed of a sequence of these simple functions, which are “a sequence of instructions” that execute in a particular order. Brann, 6:25-36, 5:15-16, Fig. 3. The instructions are used to activate a data element, indicated by the red arrows. The set of all possible individual instructions together represent the instruction set (a collection of instruction), indicated by the blue annotation. Brann describes “causing the control program to obtain from the database, upon receipt of a source message, the instructions in the database to direct the control program and call upon identified translation method programs to translate the source message to a corresponding target message.” Brann, 21:9-18. This aligns with “*each instruction of each instruction set activating a data element.*”

88. A POSITA would recognize that the transformation program, database tables and rows within those tables and above description from Brann of calling subroutines to transform a message as the claimed “*first module including a plurality of instruction sets ... activating a data element(s).*”

89. Thus, the database of Brann storing all data and instruction necessary for the translation, including the “list of subroutines to be invoked and an order in which they should be invoked,” and “the reference to find the subroutine and

execute the subroutine” renders obvious “[*the repository including...*] a first module including a plurality of instruction sets, each instruction of each instruction set activating a data element.”

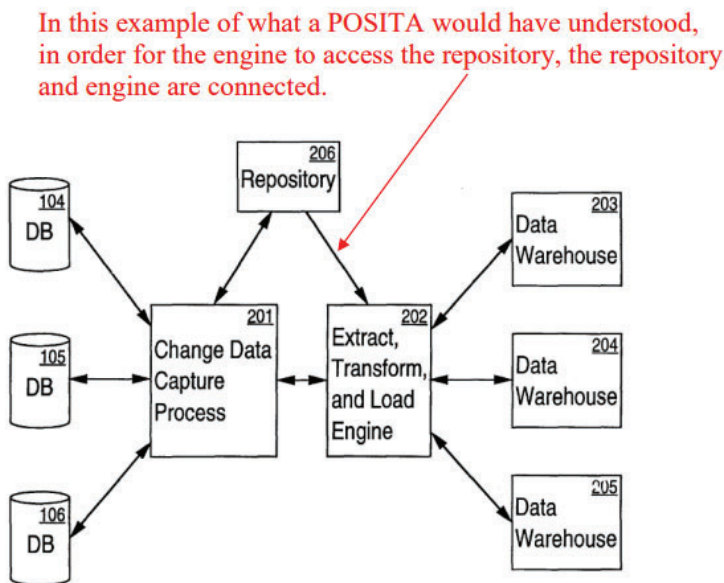
[1.3] *an active component connected to the repository having an input for loading data from the source applications, and an output for populating the destination applications with processed data, the active component processing the data from the source applications by activating data elements according to the plurality of instruction sets in the module.*

90. Brann renders this claim element obvious as it describes a control program that communicates with the database. The control program receives data from the source application via an input, transforms the data by activating subroutines according to the instructions in the database, and provides the processed data to the target application via an output.

91. Brann describes and suggests “*an active component connected to the repository*” as it teaches a control program, executing on a computer, communicating with the database. “The Control Program is a program which invokes basic computing steps in the form of Methods. It calls upon databases to tell it or instruct it as to which of these Methods should be used under which circumstances of incoming data.” Brann, 14:20-23. The control program executes on a computer. Brann, 14:34-35. “The Control Program extracts its instructions

from the database.” Brann, 7:33-34. The database is “associated with the computer.” Brann, 14:40-41. Thus Brann describes and suggests the control program executing on the computer as connected to the database.

92. To the extent that Brann does not include a figure depicting the database connected to the control program, POSITAs would have known that the database and control program would be connected. As evidence of the knowledge of POSITAs, U.S. Patent 6,014,670 teaches that POSITAs knew that the repository is connected to the transformation engine:



'670 Patent, Figure 2 (annotated)

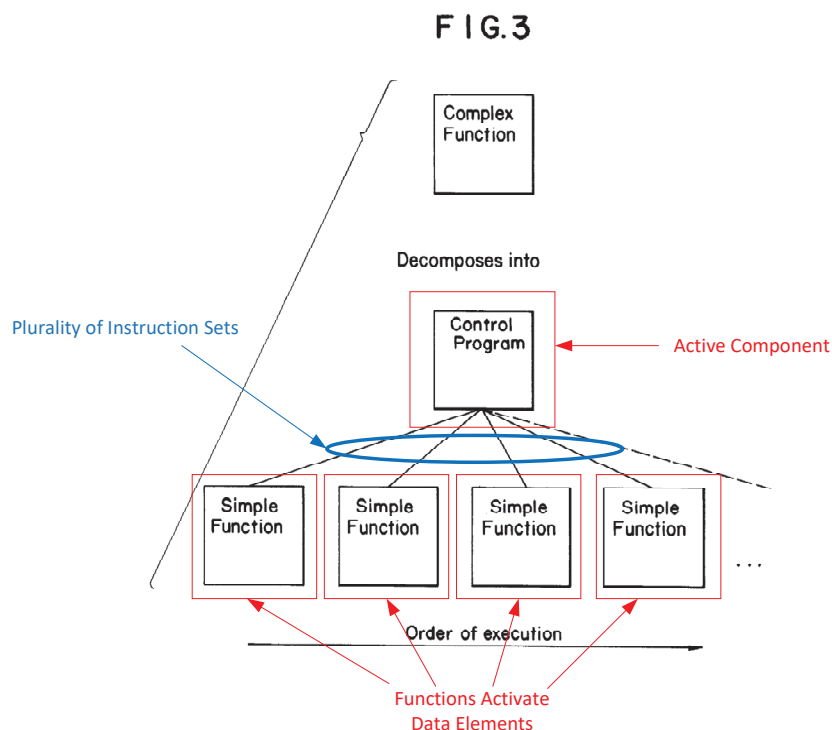
93. Brann describes and suggests the “*active component ... having an input for loading data from the source applications*” because it describes “providing the source messages to the control program.” Brann, 3:16. “The Control Program exists in a world where it is receiving stimuli. In effect it is being given

incoming data to operate upon.” Brann, 11:31-33. “[T]he Control Program receives an incoming message, a stimulus.” Brann, 12:41-42. POSITA would recognize that the control program must have an input to receive the source messages or Stimuli. A POSITA would have known that the “source” messages or Stimuli include data from the “source” application, as indicated by the name “source” message. In addition, Brann discloses a “method of efficiently translating source messages to target messages.” Brann, 2:65-68. “Once the data to be translated has been obtained (**i.e., source data**), it must be identified.” Brann, 8:6-7 (emphasis added). As such a POSITA would have known that “source messages” included “source data” from the source application, and Brann teaches “*the active component has an input for loading data from the source application.*”

94. The control program of Brann also includes “*an output for populating the destination applications with processed data*” as Brann disclosed that the transformed message (*processed data*) is provided to the target (*destination application*). “When the Control Program has executed all the Methods it is directed to do, it delivers the outgoing structure to the receiver.” Brann, 12:6-10. “The control program, responding to the database and the translation method programs, then delivers the completely translated target message to the receiver.” Brann, Abstract. A POSITA would thus recognize that Brann describes and suggests that the control program must have an output in order to populate the

destination applications with processed data.

95. Brann also describes and suggests the control program processes the data from the source message based on the instructions in the database. The instructions in the database set forth which subroutines (or methods) are to be called, and the order in which to call them, to perform the required transformation. “The Control Program extracts its instructions from the database.” Brann, 7:33-34. “The Control Program is a program which invokes basic computing steps in the form of Methods. It calls upon databases to tell it or instruct it as to which of these Methods should be used under which circumstances of incoming data.” Brann, 14:20-23. The database stores the sequence in which the subroutines are to be executed. Brann, 15:18-19.



Brann, Fig. 3 (annotated)

96. The control program controls the sequential execution of instructions from the instruction sets and simple functions based on these instructions in the database in order to process input data. Brann, 11:31-41, 6:25-36. “The Control Program then invokes the required subroutines in the required order, checking that each invocation was successful.” Brann, 7:27-30. “[C]ausing the control program to obtain from the database, upon receipt of a source message, the instructions in the database to direct the control program and call upon identified translation method programs to translate the source message to a corresponding target message.” Brann, 21:9-18.

97. In addition, Brann describes that the control program must pass information from the source application to the various subroutines in order for the message to be translated.

98. “The other information that it [control program] has to pass to the subroutine, which is parameter information, consists of a pointer to where the source field is in its temporary computer memory; where the target field is in its temporary computer memory; the size of those fields in bytes, and, if necessary, a reference to a look up table which would be used in nonalgorithmic transformations. In other words, the process of analysis populates the database in terms of the shapes and structures of messages and which ones need to be

translated and the Control Program interprets those as instructions for carrying out the translation.” Brann, 11:20-30.

99. Thus, that the control program of Brann (i) is connected to the database, (ii) has an input for loading data from the source applications, (iii) has an output for populating data in the target applications, and (iv) transforms the source’s data by activating subroutines according to the instructions stored in the database renders obvious “*an active component connected to the repository having an input for loading data from the source applications, and an output for populating the destination applications with processed data, the active component processing the data from the source applications by activating data elements according to the plurality of instruction sets in the module.*”

2. Claim 3

[3.0] The system of claim 1

100. See [1.0]-[1.3].

[3.1] *further comprising a second module including a plurality of instruction sets, each instruction of each instruction set activating a data element,*

101. As discussed in [1.2], Brann describes and suggests a database storing “a set of transformation instructions.” The set of transformation instructions forms a translation program, corresponding to the claimed “module.” Brann describes that a “multiplicity of translation programs,” that is several modules, may be

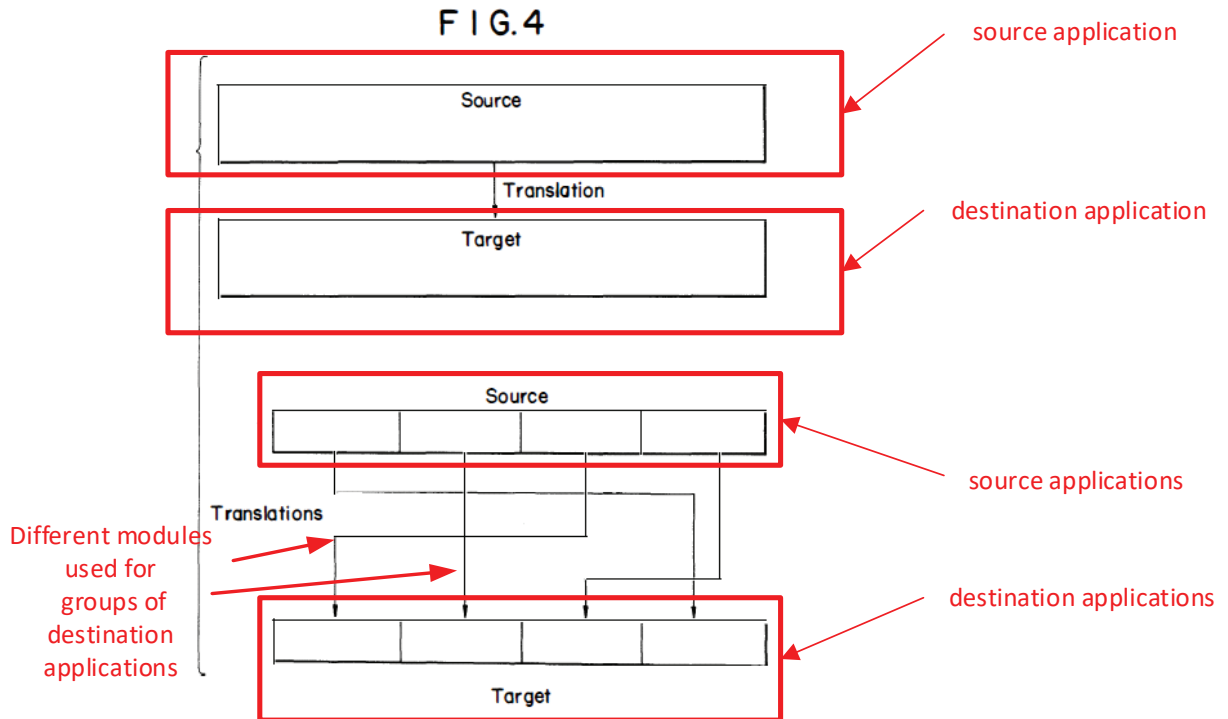
provided. Brann, 15:25-27. Brann thus describes and suggests the claimed second module.

102. In more detail, the sets of transformation instructions are entered into rows in a table in the database. Brann, 10:58 - 11:26. Each row in a database table corresponds to a different module including instruction sets activating a data element. “Each of the details of the transformation is recorded in a translation details table. The database tables are populated with this information, and the Control Program interprets this information “as instructions for carrying out the translation.” Brann, 11:26-30. A POSITA would understand the different translation programs stored in database tables, having rows storing the transformation instructions for the various translations, as a different modules including a plurality of instruction sets activating a data element. Thus, Brann teaches and suggests this claim element.

[3.2] *the first module plurality of instruction sets being defined for interface with a first group of destination applications, the second module plurality of instruction sets being defined for interface with a second group of destination applications.*

103. Brann describes and suggests that each transformation (*first module and second module*) can provide data to different targets (*a first group of destination applications and a second group of destination applications*) because

Brann's Figure 4, for example, illustrates that different transformations (*first module*) use one transformation to interface with one target (*first group of destination applications*) and another transformation (*second module*) to interface with another targets (*second group of destination applications*):



Brann, Figure 4 (annotated).

104. Brann describes that a multiplicity of translation programs may be provided for performing different translations. Brann, 15:25-28. Brann also shows that a plurality of different targets may receive the transformed messages. Brann, Fig. 4. Because Brann describes and suggests that data from a transformation can be stored in one or more targets and that different types of transformation may be

performed. Brann, 11:4-19 and 59-60. A POSITA would understand that the different transformations are required for each individual pairing of a source and a destination, as Brann also teaches that “that different computer systems represent information in different forms” and that “[i]t is well known that even in systems provided by the same manufacturer, the data structures describing objects in the real world are different in different systems, even when the real-world objects being described are identical.” Brann, 7:9-15. Thus, Brann describes implementing different transformation instructions that each interface with different groups of one or more targets, and teaches the recited limitation.

3. Claim 4

[4.0] *The system of claim 3*

105. See [3.0]-[3.2].

[4.1] *wherein at least one data element of the plurality of data elements is activated by both the first module plurality of instruction sets and the second module plurality of instruction sets.*

106. First, as discussed in [1.1], Brann describes and suggests a set of ready-made subroutines (*data elements*). Brann,6:56-65. Transformations can be created from the set of subroutines. For example, “Translation Methods are written as subroutines which receive a particular set of parameters. Each Method carries out a particular type of transformation. For example, there is one to do copying and

one to do conversion of character format numbers into integers. Each of the Methods carries out a particular type of manipulation of simple data types.” Brann, 12:47-52.

107. Brann then describes and suggests that different translation programs can be created from the subroutines. Brann also describes and suggests that the subroutines are used by multiple instruction sets. Brann further describes and suggests that different translation programs can each activate a common subroutine, particularly since the subroutines are being selected from a set of subroutines. Brann, 6:34-36; 9:10-14. A subroutine is invoked using “a reference to the subroutine code (usually the function name or a function pointer).” Brann, 8:61-65. It is well-know that subroutines “can be re-used in similar instruction needs.” Brann, 6:34-37

108. In addition, the re-used subroutine would be *activated by both the first module plurality of instruction sets and the second module plurality of instruction sets* as discussed in [1.3] and [3.1]. Specifically, Brann describes and suggests that the subroutine would be *activated* during the transformation process. Brann, 11:10-20

109. Thus, at least one subroutine of Brann that (i) is part of a set of subroutines, (ii) can be re-used in different translation programs, and (iii) is activated during the different transformations renders obvious the recited

limitation.

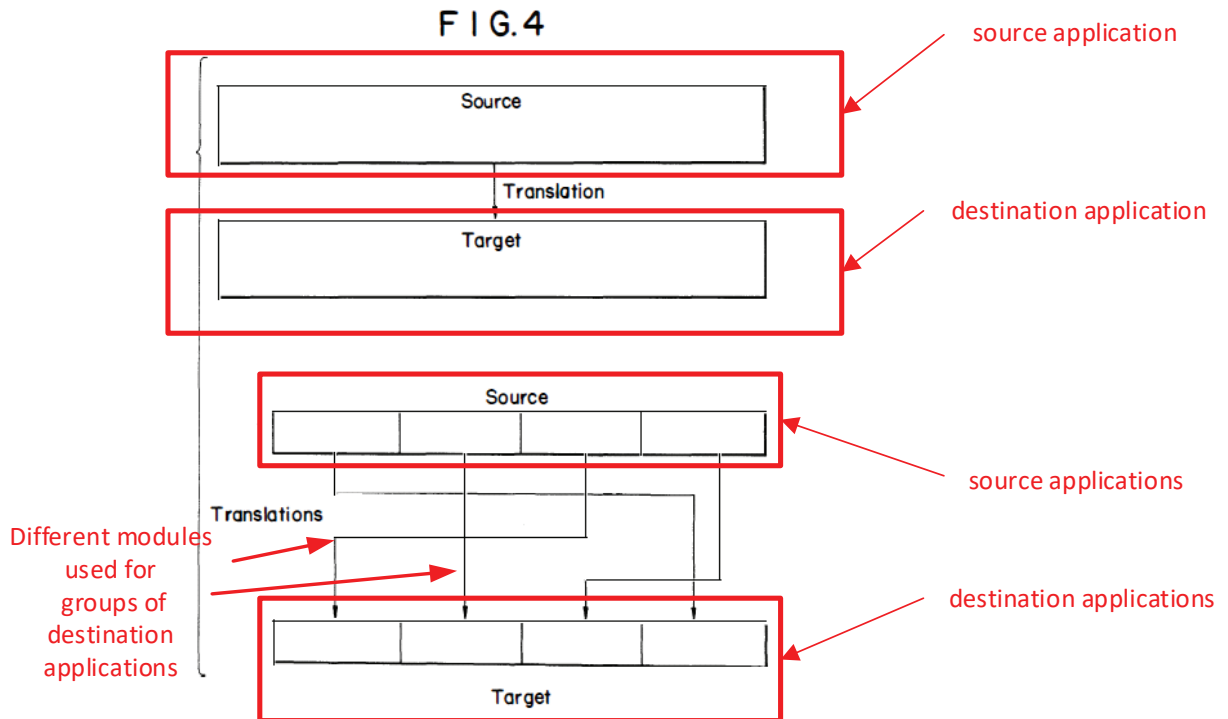
4. Claim 7

[7.0] *The system of claim 1*

110. See [1.0]-[1.3].

[7.1] *wherein the repository is capable of receiving a plurality of modules.*

111. Brann describes and suggests a “*repository of information*”, i.e. database. See [1.2]. For example, Brann discloses “storing instructions, data, and the structures of any source stimuli to be acted upon in the database.” Brann, 3:49-51. Brann illustrates a plurality of source messages may be translated for delivery to a plurality of targets. As such, different translations may be used for integrating data between different sources and targets. Brann describes that the instruction sets for these translations would be stored in different “modules” in the database:



Brann, fig. 4 (annotated)

112. Brann also describes that a multiplicity of translation programs may be provided. Brann, 15:25-28. A POSITA would understand that the database (repository) of Brann is capable of receiving a plurality of modules (multiplicity of translation programs) for the transformation of data between different sources and different targets, rendering this claim obvious.

5. Claim 8

[8.0] *The system of claim 1*

113. See [1.0]-[1.3].

[8.1] *wherein the destination applications are included within an ERP product.*

114. Brann render this limitation obvious. First, as discussed in [1.0],

Brann describes and suggests a system that integrates data between heterogeneous source and target applications.

115. Second, the '178 patent admits that ERP products were well known to a POSITA at the relevant time. '178 patent, 1:38-47. ERP products provide applications for business functions such as accounting, purchasing, and other financial matters. '178 patent, 1:38-47. Brann discusses an example of a data integration problem and financial data that is typical of ERP systems. ("An example of this is the conversion of a monetary currency code from one arbitrary form (e.g., that defined by Reuters) into another (e.g., that defined by S.W.I.F.T.)). Brann, 8:25-28. This type of financial application would typically be part of an ERP product. '178 Patent, 1:38-47. Brann also describes that one area in which the problem of data integration arises is for large scale, batch data processing tasks, such as payroll production. Brann, 1:16-30. This is an area typical of ERP systems. As such, a POSITA would have known that one example of the targets of Brann would be an application in an ERP system.

116. Accordingly, a POSITA would understand the transformation described in Brann from a monetary currency code from one arbitrary form into another form to be for the target application within an ERP product. The data transformation for financial information to targets in Brann render obvious the recited limitation.

B. Grounds 2 and 3: Claims 1 – 5, 7, 9, 10, and 23 – 25 are obvious under 35 U.S.C. §103 over Suresh alone or combined with Brann.

Grounds 2 and 3 are discussed together below. Brann is used to supplement Suresh for elements 1.0 and 23.0 (applications) and 1.2 and 23.4 (first module)..

1. Reasons to Combine Suresh with Brann

117. A POSITA would have considered the teachings of Brann and Suresh because they are analogous prior art in the same field of endeavor as the '178 patent – integrating data among multiple, disparate sources and targets. Brann, 1:10-13, 5:27-28; Suresh, 4:55-57, 6:15-19.

a.) Integrating data between applications

118. To the extent that that Suresh's Sources and Targets are not applications, it would have been obvious at the time of the alleged invention to provide applications in Suresh instead of databases or data warehouses in view of Brann. Suresh would have benefited from Brann's teachings. Any front-end applications for the databases in Suresh may be integrated via the system of Brann. For example, Brann describes the problem of communication between different types of computer programs, and provides a solution to that problem. Brann provides "an efficient system for assembling and for translating information from one computer to another." Brann, 1:10-13. A POSITA would have understood that the databases or data warehouses of Suresh could be replaced with applications

that utilize those databases to solve the problem of interoperability between different types of applications, as well as different databases. A POSITA would have recognized that companies often run different applications with back end databases and that when companies desire to partner or merge, that integrating data between such applications would have been desirable.

119. There would have been a reasonable expectation of success in the combination because integrating data between heterogeneous applications was already known. Brann, 1:10-13.

b.) Storing instructions in a database

120. To the extent that Suresh does not suggest the first module including a plurality of instruction sets, Suresh would have benefited from Brann's teachings so that the pipeline of Suresh could be quickly and easily reused or even modified for different transformations. Furthermore, adding the pipeline to the repository allows the pipeline to be more easily bound to one or more processes, and be more readily used to construct further more complex pipelines. For example, Suresh teaches that "a repository ... is used to keep track of ... mapping information relating to how data is to be mapped and transformed from target tables of the operational databases to target tables of data marts." Suresh, 6:25-29.

121. It would have been obvious to provide a first module including a plurality of instruction sets, each instruction of each instruction set activating a

data element in Suresh, as it would be use of a known technique (storing instructions) to improve known devices or methods in the same way (allowing the instructions to be modified and/or saved) to yield predictable results (calling subroutines as set forth in computer instructions, with a reasonable expectation of success. For example, it would have been obvious to do so to store and combine transformation components at run time, as taught by Brann. Brann, 3:9-15.

122. In addition, it would have been obvious to do so to eliminate the need for new programs and for manually creating the pipeline, as taught by Brann. Brann, 2:53-62. Once the pipeline is created and saved, updates may be made by revising the stored pipeline and by adding new instructions to pipelines in the database to create different transformations, as opposed to needing to create a new pipeline from scratch for every use or modification. This also allows different computers or users to use the same, stored transformation.

123. Accordingly, there would have been a reasonable expectation of success in the combination because techniques for storing transformation so that they may be quickly and easily reused or even modified techniques were already well-known and applied in data integration. Brann, 7:58-67, Suresh, 7:57-65.

1. Claim 1

[1.0] *A system for integrating data among heterogeneous source applications and destination applications, comprising:*

124. First, Suresh discloses a “computer system” used “as a part of a data warehousing system.” Suresh, 5:36-41. Figure 2, below, “illustrates a typical computer network” for implementing the system:

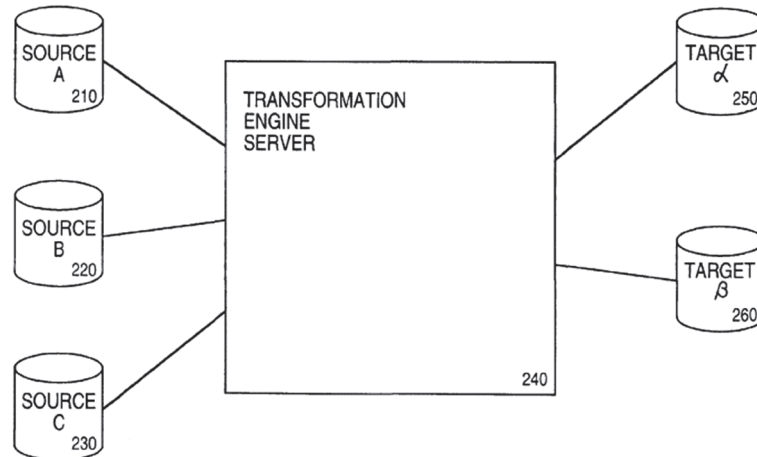


Figure 2

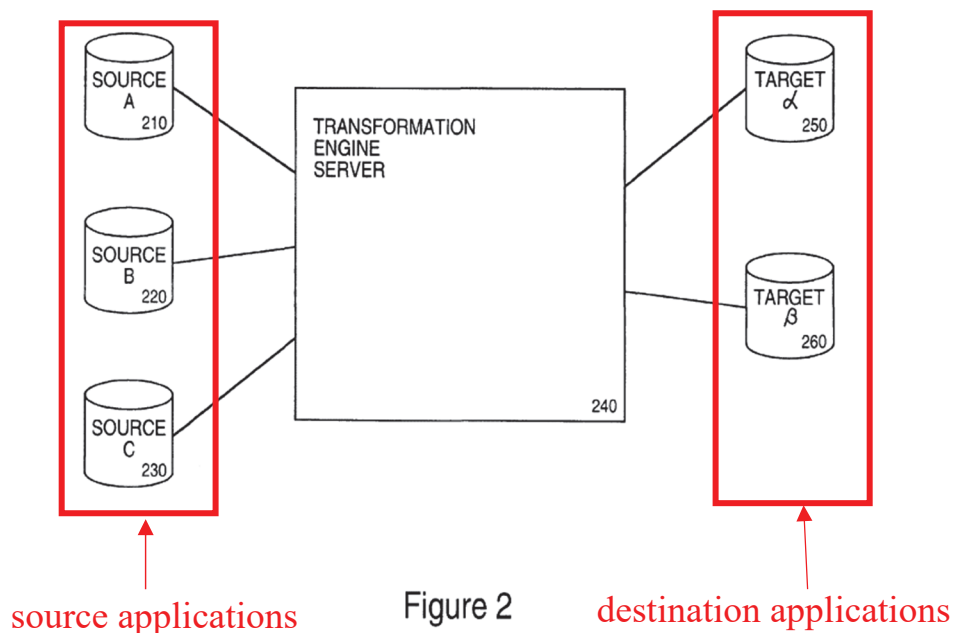
Suresh, 6:6-8, Figure 2.

125. Suresh also describes and suggests a *system for integrating data* because the system includes a “repository” and a “transformation engine server.” Suresh, 6:17, 25-26. The repository contains “mapping information relating to how data is to be mapped and transformed from target tables of the operational databases to target tables of data marts.” Suresh, 6:25-29. The transformation engine server extracts and transforms data from the source applications and loads the data into the destination applications:

In the present embodiment of the present invention, these **operational databases provide the data source from which the transformation engine server 240 can extract, transform, then load the data into the databases 250 and 260. Databases 250 and 260 are the data warehouses or data marts that are the targets of the data ETL [Extraction/Transformation/Loading] process.**

Suresh, 6:15-20.

126. Second, Suresh describes and suggests that its system integrates data *among heterogeneous source applications and destination applications*. Suresh discloses that the system comprises a transformation engine server 240 that extracts data from multiple data sources, Source A, Source B, and Source C (“*source applications*”) and loads data to multiple destination applications, Target *a* and Target *b* (“*destination applications*”):



Suresh, Figure 2 (annotated).

127. The *source applications* (Source A 210, Source B 220, and Source C 230) are *heterogeneous* because they are different and discrete “operational databases 210, 220, and 230” that “can be any of the conventional RDMS systems (such as from Oracle, Informix, Sybase, Microsoft, etc.).” Suresh, 6:8-15.

128. Similarly, the *destination applications* (Target α and Target β) are *heterogeneous* because they are different and discrete “[d]atabases 250 and 260” that are “warehouses or data marts.” Suresh, 6:19-20; *see also* Suresh, 2:27-28 (“there might exist numerous operational databases and **many different types of data marts.**”) Accordingly, because the *destination applications* are different and discrete databases such as warehouses or data marts, the *destination applications* are *heterogeneous* destination applications.

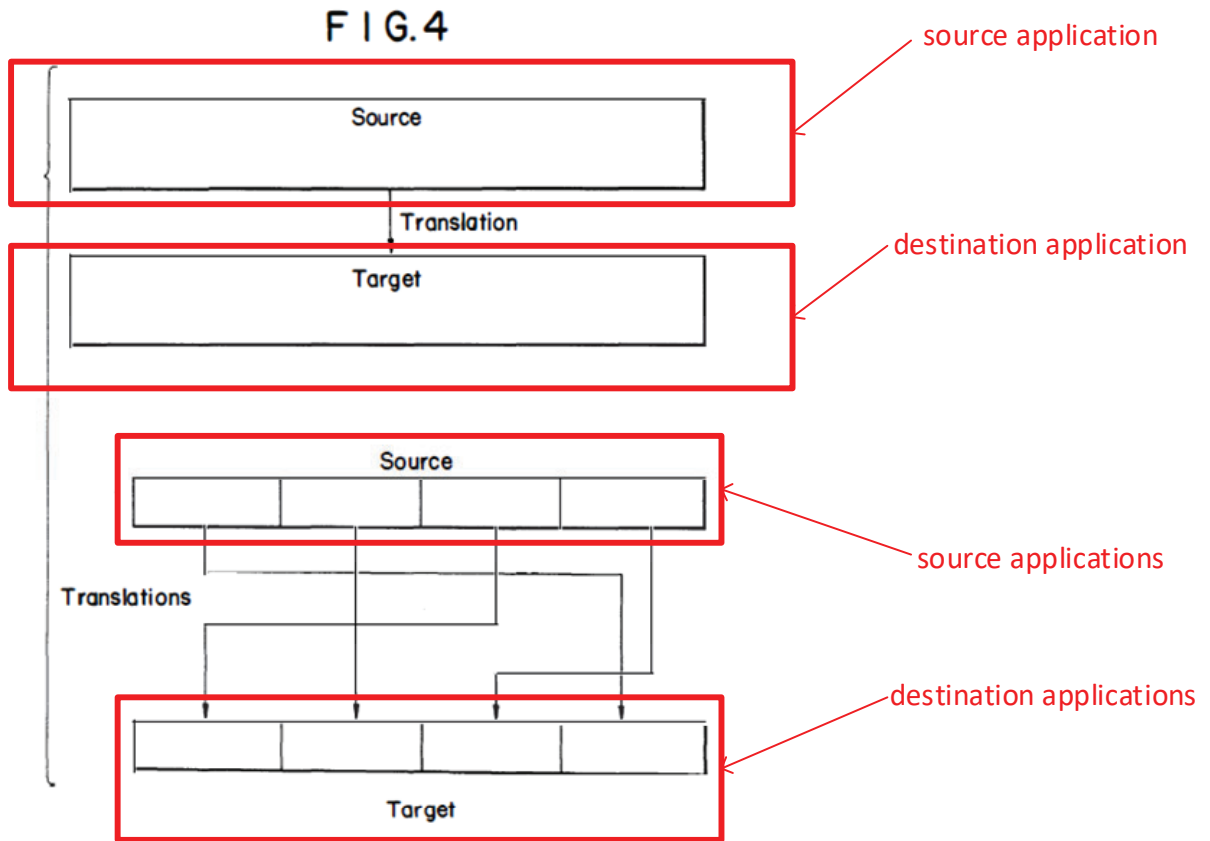
129. Additionally, Source A, Source B, and Source C (the *source applications*) are different (*heterogeneous*) than Target α and Target β (*destination applications*) because the teachings of Suresh are how to transform data from a Source to a Target. There would be no reason to transform the data if the Source and Target were the same. Accordingly, the source applications of Suresh (Source A, Source B, and Source C) are *heterogeneous* with respect to the destination applications of Suresh (Target α and Target β).

130. Further, a POSITA would recognize that Suresh’s Source A, Source

B, and Source C and Target α and Target β are *applications*. The '178 Patent itself uses “database application” and “database” interchangeably. *See, e.g.*, '178 Patent, Fig. 1, element 16 (“legacy database”) and '178 Patent, 4:7 (“legacy database applications”). A POSITA would understand that Suresh’s “conventional RDMS systems” from “Oracle, Information, Sybase, Microsoft, etc.” and “data warehouses and data marts” are *applications* because an “application” is merely a computer program designed to assist in the performance of a specific task. Microsoft Computer Dictionary, pg. 27. Correspondingly, Suresh’s systems are software applications running on computers to perform data management tasks. Suresh, 6:11-20.

131. Thus, the computer system of Suresh that extracts and transforms data from different and discrete source databases and loads the data into different and discrete target databases renders obvious “*a system for integrating data among heterogeneous source applications and destination applications,*” as claimed.

132. Brann describes a source application (source) and a destination application (target), see Fig. 4 of Brann annotated below.



Brann, fig. 4 (annotated)

133. A POSITA would have understood that the databases or data warehouses of Suresh could be replaced with applications to solve the problem of interoperability between different types of applications, as well as databases. A POSITA would have understood that data in a database or data warehouse would not be useful without an application to access the data for presentation or analysis. Combining Suresh and Brann would simply have been combining prior art elements (Suresh's database and Brann's application integration techniques) according to known methods (Brann's application integration) to yield predictable

results (integrated data among different types of applications).

134. There would have been a reasonable expectation of success in the combination. This is because a system for integrating data between heterogeneous applications was already known. Brann, 1:10-13. Further, techniques for integrating data into a heterogenous application, such as “Enterprise Application Integration” and “Enterprise Resource Planning,” were well known. ’178 patent, 1:5-8, 1:38-43. Numerous products already in use provided the same functionality without the need for custom code at the front end or at the back end. ’178 patent, 1:41-51.

135. Thus, the computer system of Suresh that extracts and transforms data from different source database systems and loads the data into different target database systems, alone or in combination with the teachings of Brann, renders obvious “*a system for integrating data among heterogeneous source applications and destination applications.*”

[1.1] *a repository of information relating to the source applications and the destination applications, the repository including a plurality of data elements each being capable of performing a discrete operation on a piece of data, and*

136. First, as discussed in [1.0], Suresh discloses *a repository*. See Suresh, 6:23-29 (“a repository, which is used to keep track of ... mapping information”).

137. Second, Suresh describes and suggests that the repository contains

information relating to the source applications and the destination applications because the repository stores mapping information that relates to how data is to be transformed from the tables of the operational databases (e.g., *information relating to the source applications*) to the tables of the data warehouses and data marts (e.g., *information relating to the destination applications*):

A client is used to create **a repository, which is used to keep track of session information as well as mapping information relating to how data is to be mapped and transformed from target tables of the operational databases to target tables of data marts.**

Suresh, 6:25-29.

138. Third, Suresh describes and suggests that the mapping information in the repository further includes a *plurality of data elements each being capable of performing a discrete operation on a piece of data* because Suresh describes and suggests that one example of the mapping information in the repository are “transformation components,” (“*a plurality of data elements*”) each of which is responsible for one specific type of transformation:

Instead of writing one block of source code that is responsible for all phases of the data ETL process, a more disciplined and architectural approach is taken in packing the source code such that **the responsibility is distribute to smaller units (i.e., components) of source code. Each one of these software components is responsible for one specific type of transformation.**

Suresh, 6:46-52.

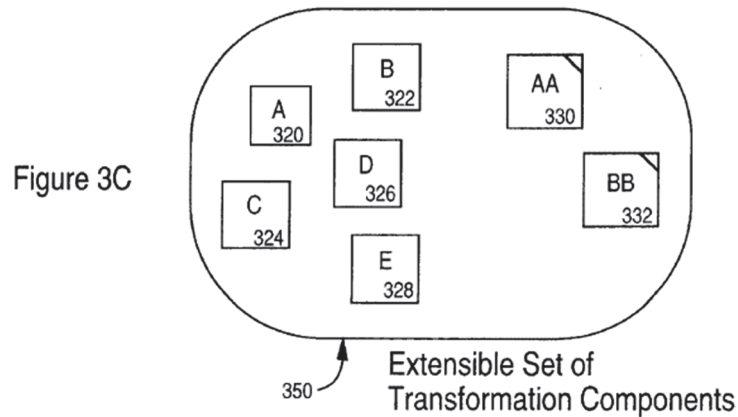
This extensible set 350 of transformation components can be thought of as a pool (i.e., **a base of ready-made transformation components** from which the users can draw on to build functionally more sophisticated transformations in the data ETL process).

Suresh, 6:61-65.

A **transformation component** is the same as a transformation object, which is **a software binary file acting as an individual unit that possesses built-in autonomy** and encapsulates the functionalities of a transformation.

Suresh, 5:18-21.

139. In fact, “there are thirteen different transformation components: source, target, expression, aggregation, filter, rank, update strategy, sequence, joiner, lookup, stored procedure, external procedure, and normalizer.” Suresh, 10:35-11:6. Stored procedures can be programmed by a user or developer to perform discrete operations on data. Suresh, 6:51-55. Suresh’s Figure 3C, depicted below, also illustrates the discrete transformation components:



Suresh, Figure 3C.

140. Thus, the repository of Suresh that stores mapping information that (i) relates to how to transform data from the source to the destination and (ii) includes transformation components that are each responsible for a specific type of transformation renders obvious “*a repository of information relating to the source applications and the destination applications, the repository including a plurality of data elements each being capable of performing a discrete operation on a piece of data,*” as claimed.

[1.2] *[the repository including...] a first module including a plurality of instruction sets, each instruction of each instruction set activating a data element; and*

141. First, Suresh uses *a first module* because it describes a “pipeline” built with transformation components:

FIG. 4 illustrates the pipeline-based aspect of the transformation architecture through depicting one particular embodiment of the present invention. Again, the extensible set 400 of transformation components is shown here as a repository of transformation components available for use to build functionally more sophisticated transformations. Arrow 410 indicates the step taken by the users and enabled by the componentized aspect of the architecture to manually form a new transformation that consists of a cascading sequence of transformation components. For example, **a pipeline can include a source table 420, an expression transformation 422, an aggregation transformation 424, and a target table 426.**

Suresh, 7:3-15.

142. Suresh describes and suggests that the pipeline is built by a user to add functionality to the data transformation process:

This extensible set 350 of transformation components can be thought of as a pool (i.e., **a base of ready-made transformation components** from which the users can draw on to build functionally more sophisticated transformations in the data ETL process).

Suresh, 6:61-65.

And, one example of a pipeline configured to operate with the transformation engine server is depicted in Suresh's Figure 4, below:

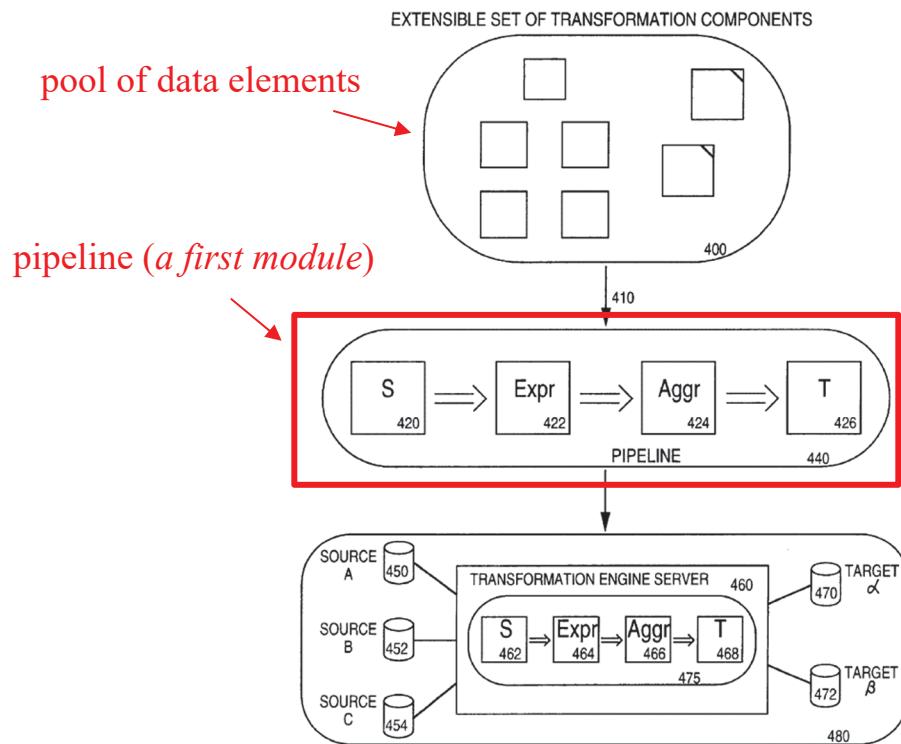
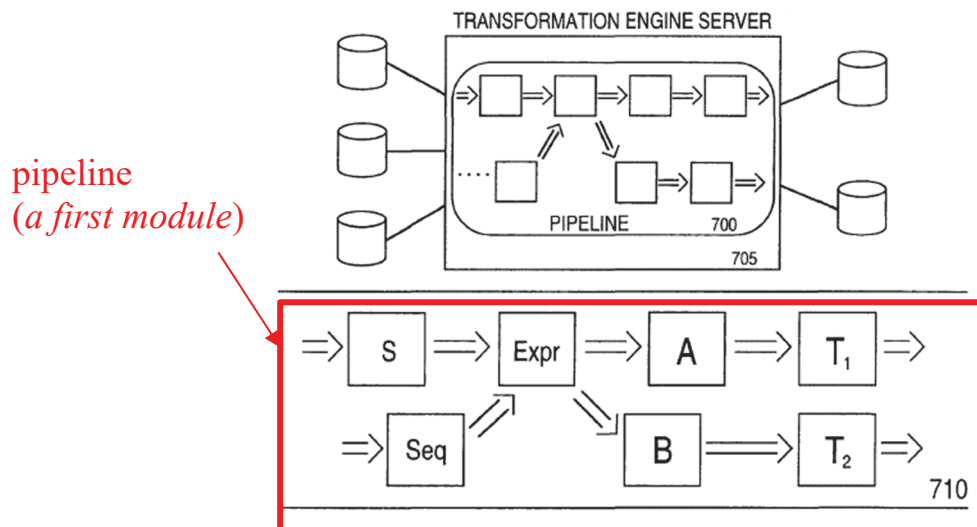


Figure 4

Suresh, Figure 4 (annotated)

143. Another example of a pipeline configured to operate with the transformation engine server is depicted in Suresh's Figure 7, below:



Suresh, Figure 7 (in part), (annotated).

144. A pipeline in Figure 4 or Figure 7 is a *first module*.

145. Second, Suresh describes and suggests that a pipeline includes a *plurality of instruction sets* because a pipeline includes cascading sequences of transformation components:

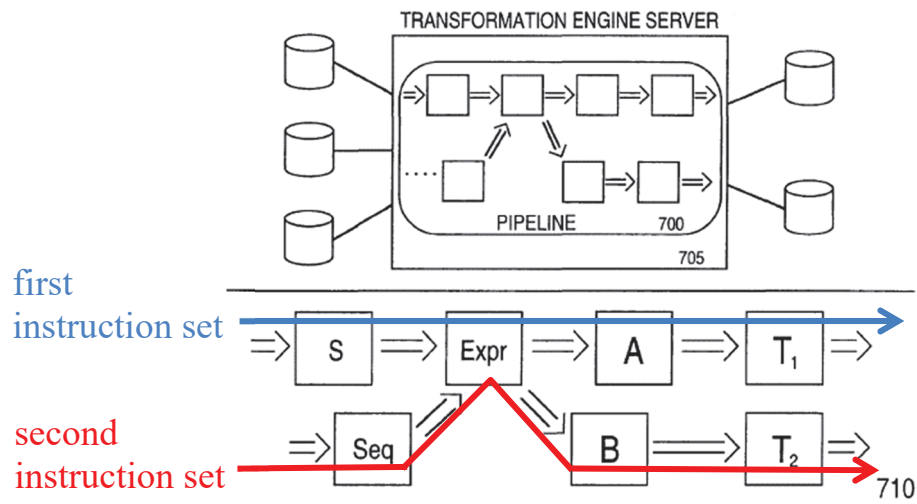
This **cascading sequence of transformation components exemplifies the pipeline-base** aspect of the transformation architecture.

Suresh, 7:25-27.

For example, **pipeline 700** of transformation engine server 705 depicts one single pipeline of transformation components. More specifically, pipeline 710 shows each of the transformations. **A single transformation can accept one or more sets of inputs from one or more sources and/or prior transformations and can output one or more data sets to one or more subsequent transformations and/or targets.**

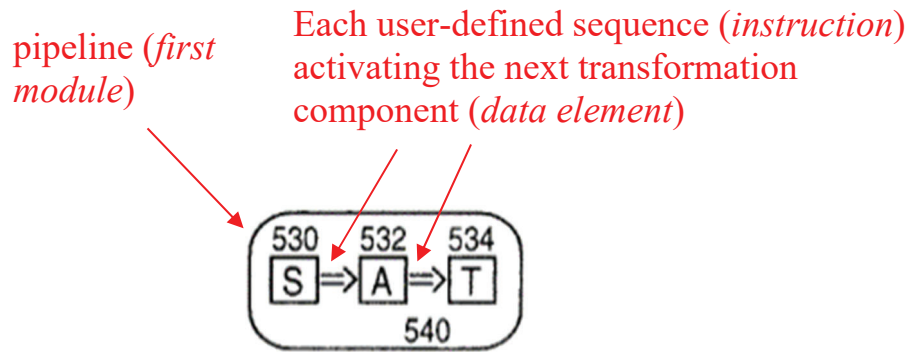
Suresh, 8:27-34.

146. Suresh's Figure 7 depicts two example cascade sequences ("*plurality of instruction sets*") included in the pipeline:



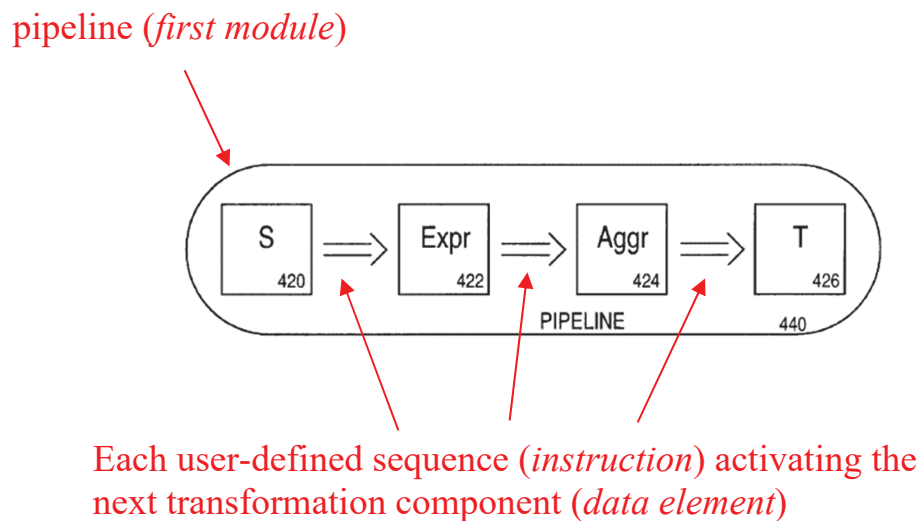
Suresh, Figure 7 (in part) (annotated).

147. Third, Suresh describes and suggests the cascading sequence (“*instruction set*”) of transformation components (“*data elements*”) with where the user-defined sequence activates the next transformation component because the “transformation components that are being formed into the pipeline” are “coupled together (e.g. in a Lego-like fashion).” Suresh, 7:38-42. The “user also manually arranges the selected transformation components to form one or more sequential and/or parallel pipeline(s).” Suresh, 9:8-10. A pipeline with transformation components that are coupled together is depicted in Figure 5, below:



Suresh, Figure 5 (in part), (annotated)

148. Suresh’s Figure 4 also depicts a “cascading sequence of transformation components” where “[t]he expression transformation 422 performs a calculation based on values within a single record from source table 420” and activates using an *instruction* from the aggregate transformation components 424. Suresh, 7:3-37. The aggregate transformation component, in turn uses another *instruction* to store a result in target table 426. Suresh, 7:3-37.

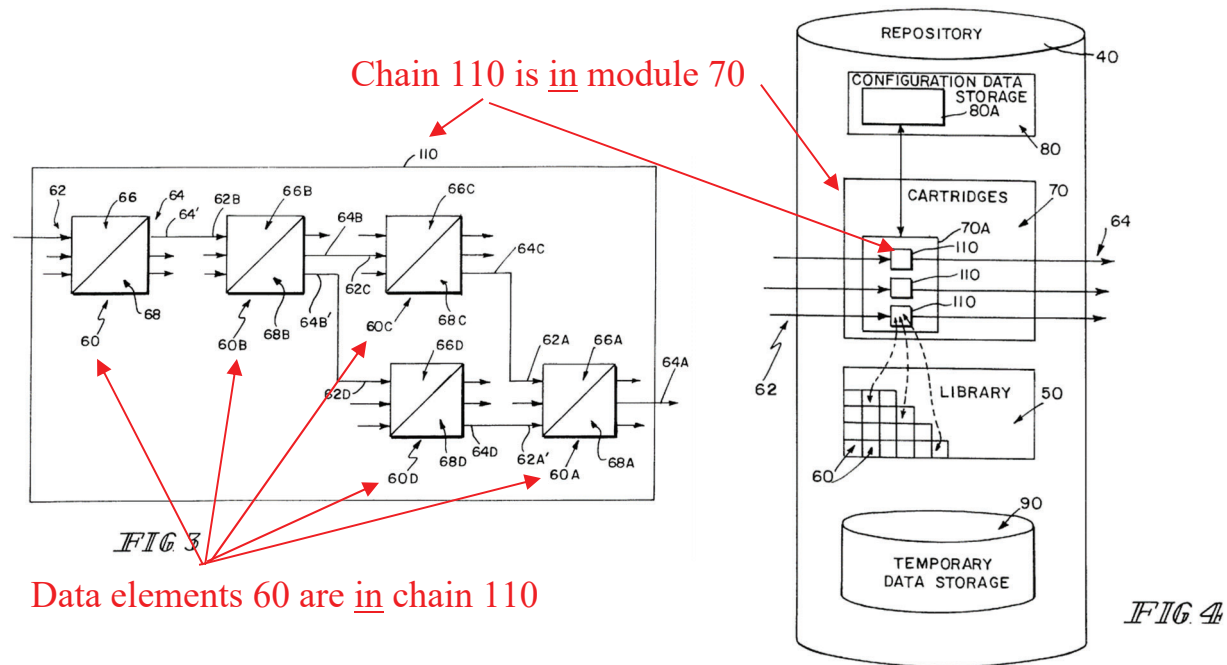


Suresh, Figure 4, (in part) (annotated)

149. Fourth, Suresh teaches that the *repository* includes the pipeline (“*first*

module”). The repository keeps track of “mapping information relating to how data is to be mapped and transformed from target tables of the operational databases to target tables of data marts.” Suresh, 6:23-29. A pipeline is a “transformation that consists of a cascading sequence of transformation components,” that is how data is being “mapped and transformed.” Suresh, 7:9-12, 6:27-28. Accordingly, because the repository stores the information relating to how data is transformed and the pipeline represents the specific sequence for how to transform the data, a POSITA would understand that the pipeline is part of the mapping information stored by the repository.

150. If Patent Owner argues that the recited “module” cannot contain “data elements” by citing to column 7, lines 38-41, Patent Owner’s argument ignores the rest of the specification. The specification describes that the data elements are “assembled into combinations of data elements (data element chains 110),” which may be inside the module. ’178 Patent, 6:36-38.



'178 Patent, Figure 3 and 4 (annotated)

151. Thus, the repository of Suresh that includes mapping information, such as pipelines having a plurality of cascading sequences, where each cascading sequence is a series of transformation components that execute in sequence in operation with the transformation engine server to add functionality renders obvious *“a repository including ... a first module including a plurality of instruction sets, each instruction of each instruction set activating a data element.”*

152. To the extent that this claim element is not disclosed by Suresh, it would be obvious to a POSITA to provide a first module including a plurality of instruction sets, each instruction of each instruction set activating a data element, for example as disclosed in Brann.

153. Brann describes a repository (database) that includes data structures,

for example, a first module (database tables) that store instruction sets for activating data structures (subroutines). The programs described in Brann include two parts - a Control Program and subroutines, which carry out the required program steps. The purpose of the Control Program is to receive the incoming stimuli, with their associated data, and derive the appropriate action to be taken. The action takes the form of a list of subroutines to be invoked and an order in which they should be invoked. The Control Program then invokes the required subroutines in the required order, checking that each invocation was successful. The Control Program extracts its instructions from the database and retains them, in the form of arrays of data structures, in the computer's memory. Brann, 7:21-35.

154. The database of Brann stores a set of transformation instructions which includes the originating data elements (i.e., a field in the first structure), a resulting data element (i.e., a field in the second structure), and the Transformation Method required between them. Each one of those makes up one row in a table of the database. Brann, 10:58 – 11:10. A method table in the database stores a list of subroutines to be invoked and the order in which they should be invoked. Brann, 7:23-39; 11:10-23. Each instruction activates its associated subroutine. Brann, 7:21-35. The method table includes pointers to the particular subroutines. A function reference is used to activate the selected subroutine. Brann, 7:31-47; See also Brann, 11:10-19. Brann thus describes and suggests “*a first module including*

a plurality of instruction sets.” Annotated figure 5 of Suresh below illustrate how the “first module” of Brann would be included therein. The “instruction sets” define the order in which the transformation components from the extensible set are to be executed. The instruction sets are included in the first module, which would be logically stored in the same repository as the extensible set of transformation components.

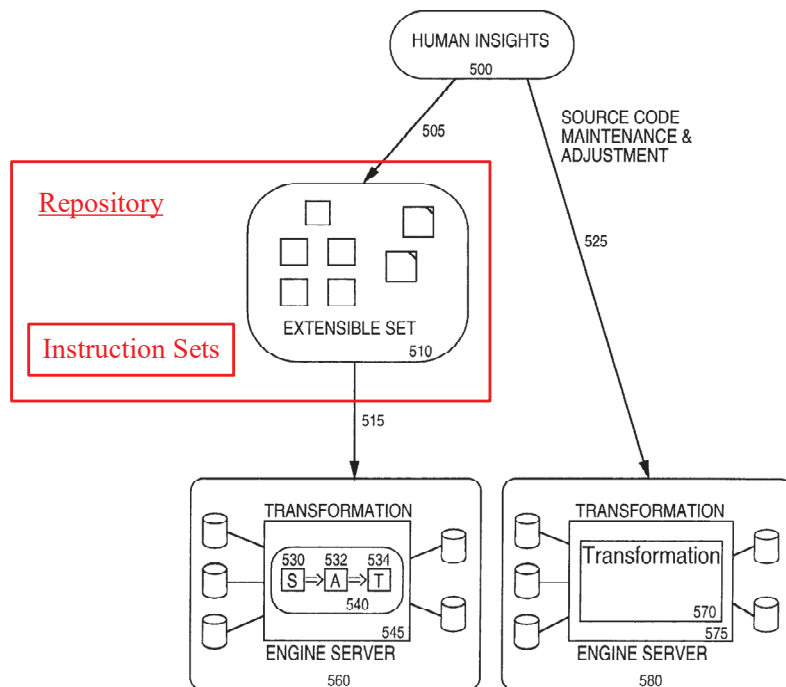


Figure 5

Brann, fig. 5 (annotated)

155. Once a pipeline, that is the sequence in which the transformation are to be called, is created, it would be stored in the repository. A POSITA would have understood that doing so would allow the pipeline to be quickly and easily reused

or even modified for different transformations.

156. Thus, the repository of Suresh that includes mapping information, such as pipelines having a plurality of cascading sequences, where each cascading sequence defines a series of transformation components to be execute in sequence in operation with the transformation engine server to add functionality combined with the teachings of Brann of storing sets of transformation instructions in a database renders obvious “*a repository including ... a first module including a plurality of instruction sets, each instruction of each instruction set activating a data element.*”

[1.3] *an active component connected to the repository having an input for loading data from the source applications, and an output for populating the destination applications with processed data, the active component processing the data from the source applications by activating data elements according to the plurality of instruction sets in the module.*

157. First, Suresh describes and suggests a transformation engine server 240 which is *an active component* because it can extract, transform, and load data:

In the present embodiment of the present invention, these operational databases provide the data source from which **the transformation engine server 240 can extract, transform, then load the data into the databases 250 and 260.** Databases 250 and 260 are the data warehouses or data marts that are the targets of the data ETL process.

Suresh, 6:15-20.

158. The transformation engine server is depicted in Figure 2, below:

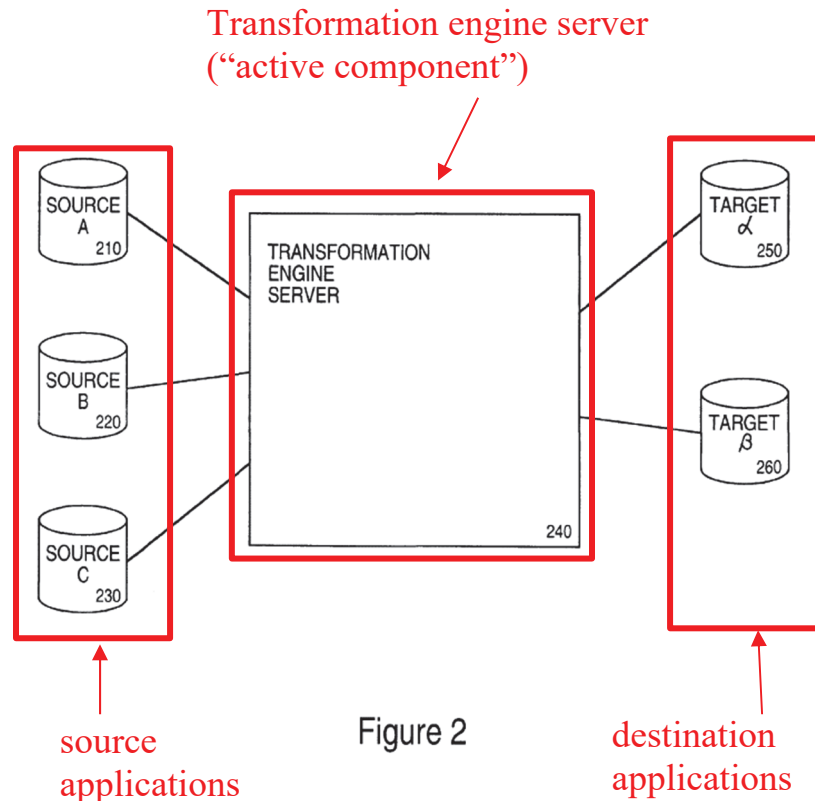


Figure 2

Suresh, Figure 2 (annotated).

159. Second, a POSITA would understand that Suresh describes and suggests that the transformation engine server (“*active component*”) is connected to the *repository* because the mapping information in the repository (including the pipeline and its transformation components) is accessed by the transformation engine server to perform the pipelined transformations within the engine.

Again, the extensible set 400 of transformation components is shown here as a repository of transformation components available for use to build functionally more sophisticated transformations.

...

This cascading sequence of transformation components exemplifies the pipeline-base aspect of the transformation architecture. **The pipelined transformations exist within the transformation engine server 460.**

Suresh, 7:3-28.

A client is used to create a **repository, which is used to keep track of session information as well as mapping information relating to how data is to be mapped and transformed from target tables of the operational databases to target tables of data marts.**

Suresh, 6:23-29.

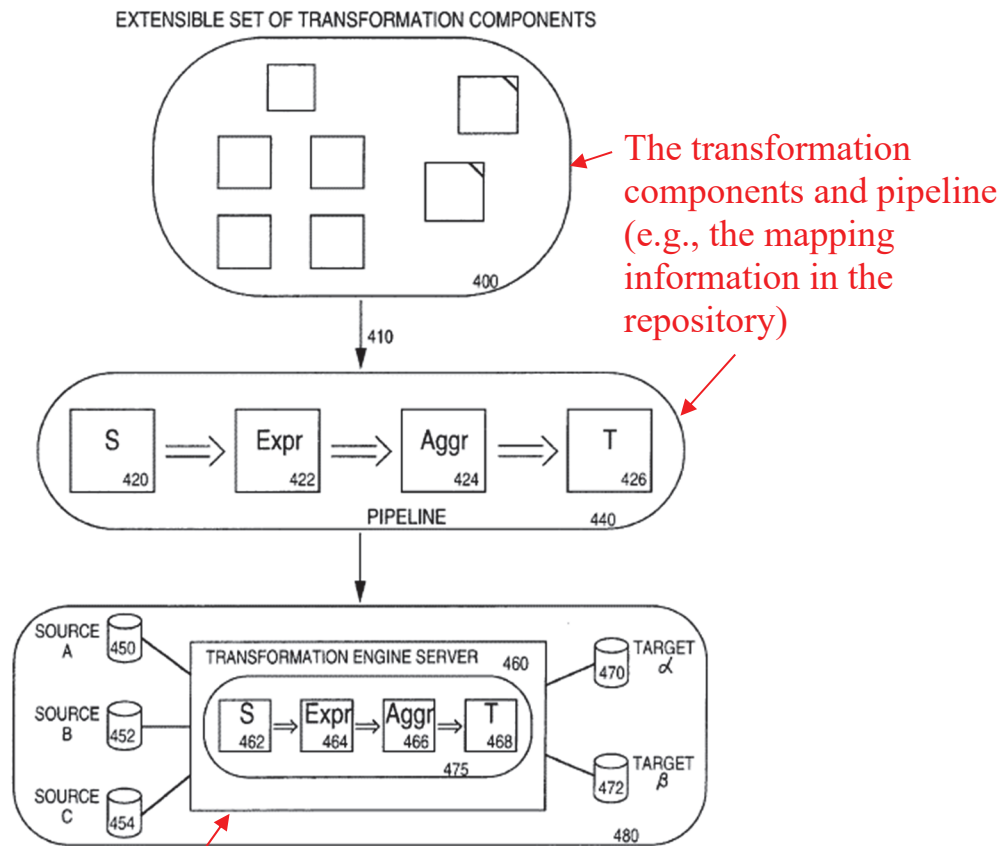


Figure 4

The pipeline and transformation components (e.g., the mapping information from the repository) being executed by the transformation engine server as a pipelined transformation

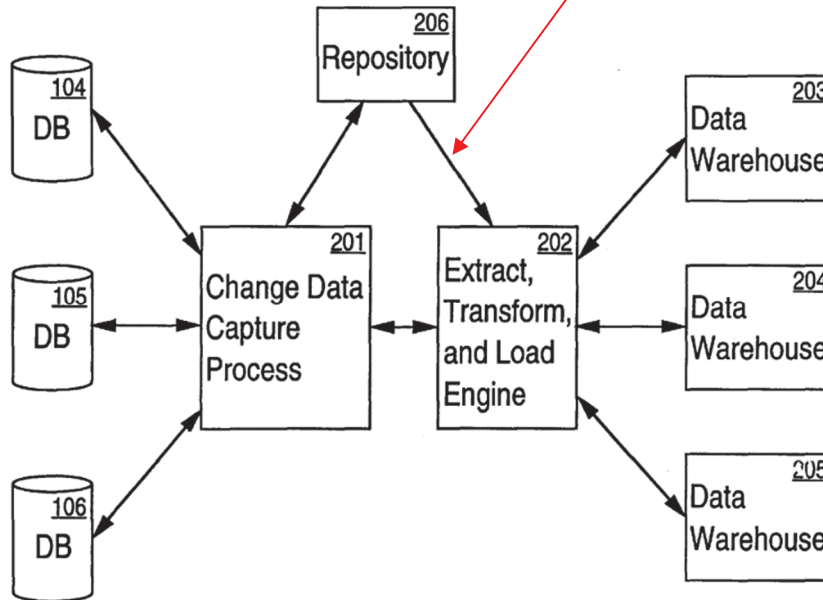
Suresh, Figure 4 (annotated).

160. Suresh's Figure 4 illustrates a set of transformation components 400 and a pipeline 440 that includes transformation components 420, 422, 424, and 426 selected into the pipeline 440. The set of transformation components 400 and the pipeline 440 are stored in the repository as "mapping information relating to how data is to be mapped and transformed from target tables of the operational databases to target tables of data marts." Suresh, 6:25-29. Because the "pipelined

transformations exist within the transformation engine server 460,” the transformation engine server 460 would access pipeline 440 from the repository, install pipeline 440 within the transformation engine server 460 (shown as pipeline 475 in Figure 4), and “execute the pipeline” using the “pipelined transformations.” Suresh, 9:13-38, 7:27-28.

161. To the extent that Suresh does not include a Figure explicitly depicting that the repository is connected to the transformation engine server, POSITAs would have known that the repository and transformation engine would be connected because the information in the repository needs transferred to the transformation engine to perform the transformation. Also, U.S. Patent 6,014,670, assigned to the same assignee as Suresh, teaches that “repository 206 is used to store the requisite session and mapping information used by the extracting, transformation, and loading process 202,” (Zamanian, 4:31-35) and shows that the repository is connected to the transformation engine:

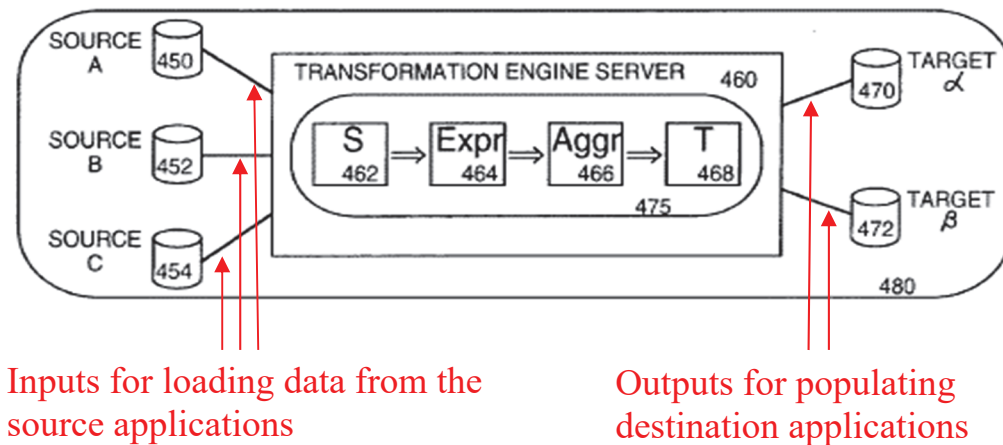
In this example of what a POSITA would have understood, in order for the engine to access the repository, the repository and engine are connected.



Zamanian, Figure 2 (annotated).

162. Third, Suresh describes and suggests that the transformation engine has an input for loading data from the source applications, and an output for populating the destination applications with processed data, because “[d]ata from any of the source tables 450, 452, and 454 can be transformed by the pipelined transformations within transformation engine server 460 and stored in either and/or both of target tables 470 and 472.” Suresh, 7:28-32, *see also id.*, 6:25-29 (“data is to be mapped and transformed from **target tables of the operational databases to target tables of data marts.**”) Suresh’s Figure 4, depicted below, also illustrates *input[s] for loading data from the source applications and output[s] for populating*

the destination applications with processed data, as claimed.



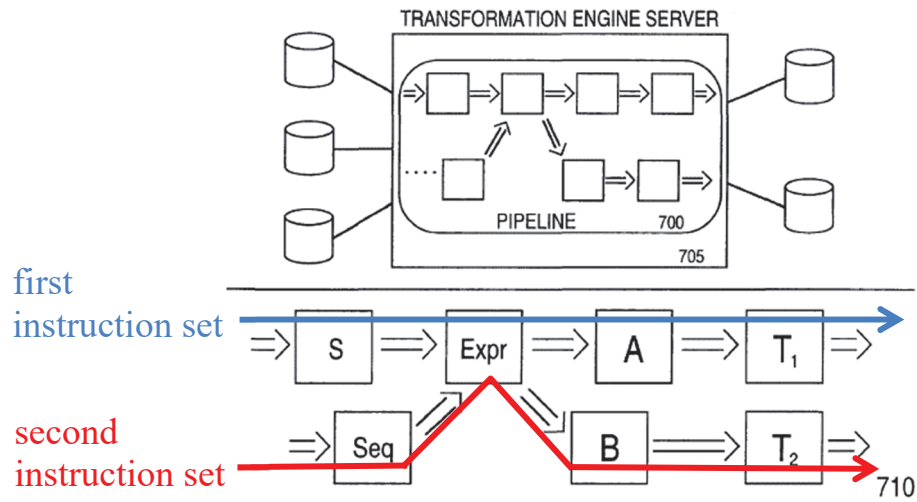
Suresh, Figure 4 (in part) (annotated).

163. Fourth, Suresh describes and suggests *the active component processing the data from the source applications by activating data elements according to the plurality of instruction sets in the module* because the transformation engine server (*active component*) transforms data from the source applications activating the sequence of transformation components (“*activating data elements according to plurality of instruction sets*”) in the pipeline (“*module*”) as a pipelined transformation:

The pipelined transformations exist within the transformation engine server 460. **Data from any of the source tables 450, 452, and 454 can be transformed by the pipelined transformations within transformation engine server 460** and stored in either and/or both of target tables 470 and 472.

Suresh, 7:27-32.

164. The transformation engine server processing data by activating different transformation components according to the cascading sequences in the pipeline is also depicted below:



Suresh, Figure 7 (in part) (annotated).

Thus, the transformation server of Suresh that (i) is connected to the repository of transformation components, (ii) has an input for loading data from the source applications, (iii) has an output for populating data in the target applications, and (iv) transforms the source's data by activating transformation components according to cascading sequences in the pipeline renders obvious "*an active component connected to the repository having an input for loading data from the source applications, and an output for populating the destination applications with processed data, the active component processing the data from the source applications by activating data elements according to the plurality of instruction*

sets in the module.”

2. Claim 2

[2.0] *The system of claim 1*

165. See [1.0]-[1.2].

[2.1] *wherein the plurality of instruction sets are predefined to facilitate interface with a particular destination application.*

166. First, Suresh describes and suggests the cascading sequences (“plurality of instruction sets”) of transformation components are defined to facilitate interface with a particular target table of a data mart (“a particular destination application”) because “data is to be mapped and transformed from target tables of the operational databases to target tables of data marts.” Suresh, 6:25-29; see also Suresh 7:9-32 (“[d]ata from any of the source tables 450, 452, and 454 can be transformed by the pipelined transformations within transformation engine server 460 and stored in either and/or both of target tables 470 and 472.”)

167. Second, Suresh describes and suggests that the cascading sequences (“plurality of instruction sets”) are predefined because “the step taken by the users and enabled by the componentized aspect of the architecture to manually form a new transformation that consists of a cascading sequence of transformation components,” before the transformation server engine performs the pipelined transformation on the sequence. Suresh, 7:9-12, 7:25-27; see also Suresh, 7:27-32

(“Data from any of the source tables 450, 452, and 454 can be transformed by the pipelined transformations within transformation engine server 460 and stored in either and/or both of target tables 470 and 472.”)

168. Thus, the cascading sequences of Suresh that (i) interface with the target table in a data mart and (ii) are defined by users and are stored in the repository before the pipelined transformation executes renders obvious “wherein the plurality of instruction sets are predefined to facilitate interface with a particular destination application,” as claimed.

3. Claim 3

[3.0] *The system of claim 1*

169. See [1.0]-[1.2].

[3.1] *further comprising a second module including a plurality of instruction sets, each instruction of each instruction set activating a data element,*

170. First, as discussed in [1.2], Suresh describes and suggests a pipeline that includes cascading sequences (“*instruction sets*”) where each instruction in the sequence activates the next transformation component (“*data element*”).

171. Second, Suresh describes and suggests multiple pipelines (“*modules*”). Any of the pipelines, such as pipeline 630 can be a second pipeline (“*second module*”) with cascading sequences (“*plurality of instruction sets*”):

FIG. 6 illustrates the pipeline-based aspect of the architecture and **the level of pipelines of transformation components**. An exemplary arrangement of transformation components that the users have selected from the extensible set of transformation components is shown.

Suresh, 7:66-8:3.

Transformation system 600 shows that three independent pipelines have naturally emerged. Each of the separate processes 630, 640, and 650 marks (separates) out the natural boundary of an independent task to be executed by the transformation engine server.

Suresh, 8:5-9.

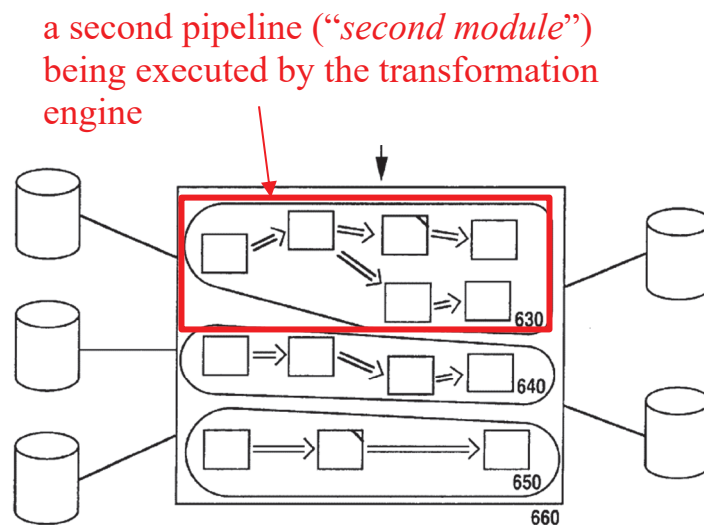


Figure 6

Suresh, Figure 6 (in part) (annotated).

172. Brann describes the sequences in which the transformation components are activated, that is the “pipeline(s),” may include a multiplicity of

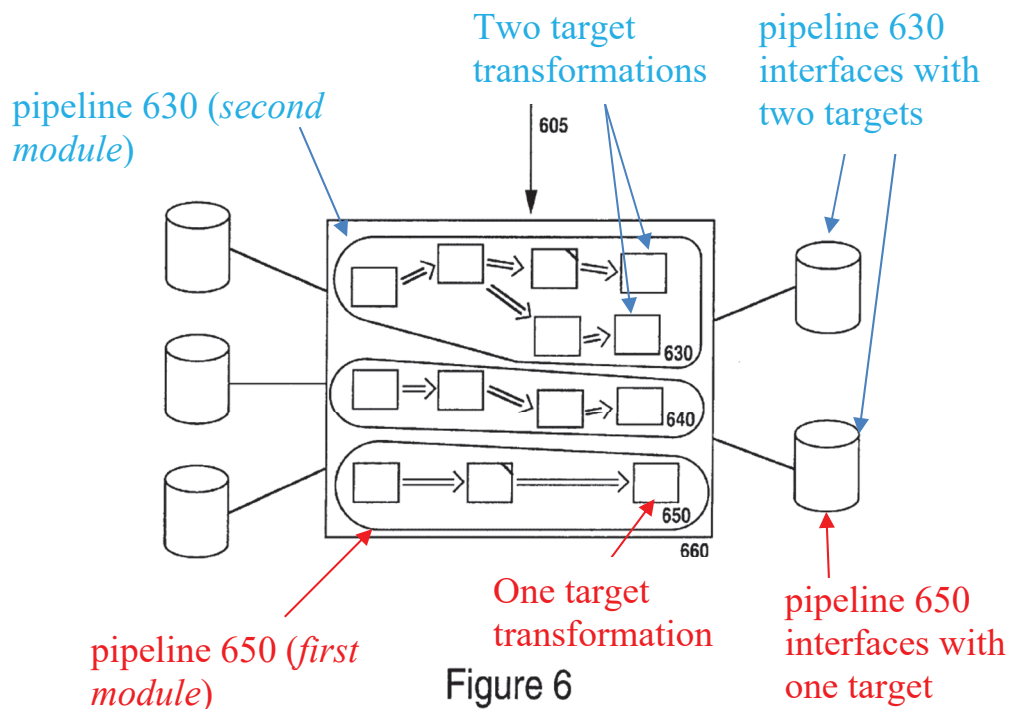
sets of instructions. See [1.2] The second pipeline of Suresh would have been modified in the same way as the first pipeline. *Id.* A POSITA would have combined this teaching of Brann with Suresh. *Id.* Thus, a second pipeline of Suresh that includes cascading sequences of instructions to activate transformation components renders obvious the recited limitation, alone or in view of Brann.

[3.2] *the first module plurality of instruction sets being defined for interface with a first group of destination applications, the second module plurality of instruction sets being defined for interface with a second group of destination applications.*

173. Suresh describes and suggests that each pipeline (“*first module and second module*”) can push data to different targets (“*a first group of destination applications and a second group of destination applications*”) because each pipeline “**can output one or more data sets to one or more subsequent transformations and/or targets.**” Suresh, 8:31-35; *see also id.*, 7:27-37 (“[d]ata from any of the source tables 450, 452, and 454 can be transformed by the pipelined transformations within transformation engine server 460 and **stored in either and/or both of target tables 470 and 472. Different transformations can be added...**”). The pipelines each interface with the target(s) using a “target transformation” that “receive[s] data from other transformations” and “then make[s] the data available to the target database.” Suresh, 10:41-44. Figure 7 and

its corresponding description further show that the transformation components T_1 and T_2 push data to the targets. *See*, Suresh, Figure 7, 8:40-9:3.

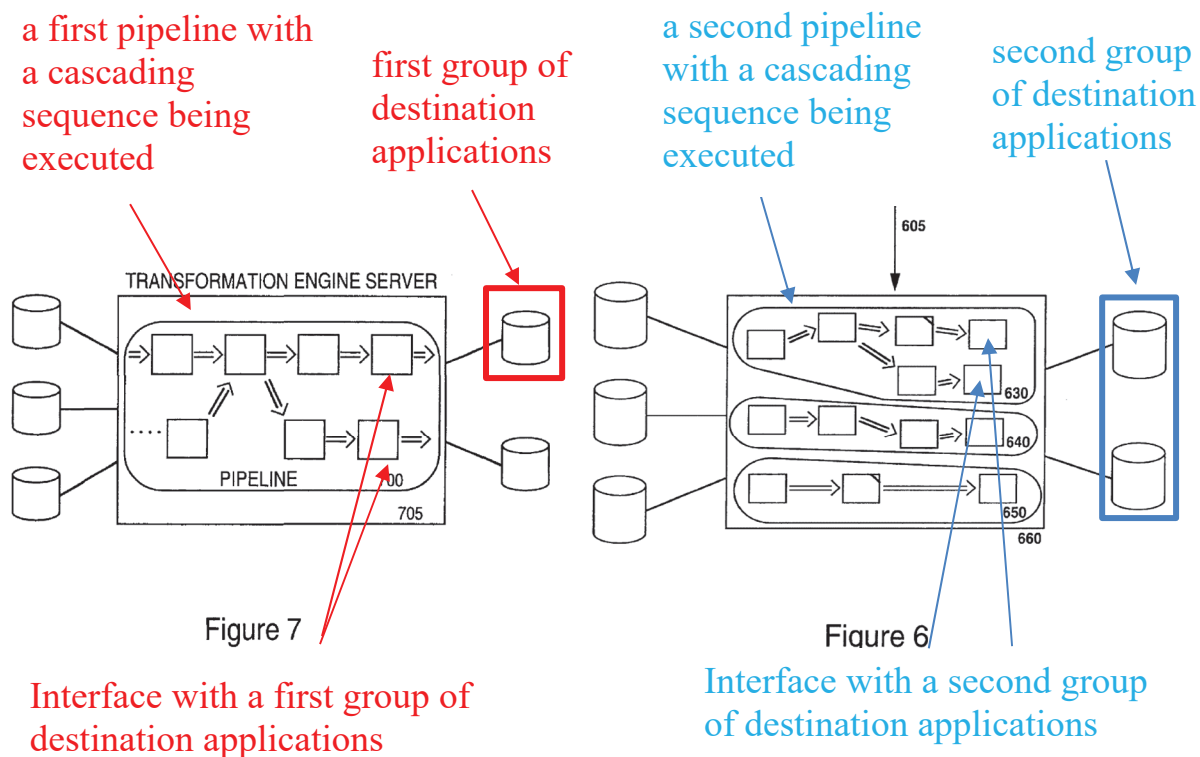
174. Suresh's Figure 6, for example, illustrates that pipeline 650 (*first module*) uses one target transformation to interface with one target database (*first group of destination applications*) and pipeline 630 (*second module*) interfaces with two target tables that map to two target databases (*second group of destination applications*):



Suresh, Figure 6 (in part) (annotated).

175. Similarly, because Suresh describes and suggests that data from source tables can be stored in different target tables (Suresh, 8:31-35), a POSITA

would have found it obvious to implement a pipeline 700 (“*first module*”) with its cascading sequences (“*first module plurality of instruction sets*”) interfacing with a first group of target databases and pipeline 650 with its cascading sequences (“*second module plurality of instruction sets*”) interfacing with a second group of target databases. This embodiment is depicted below:



Suresh, Figures 6 and 7 (in part) (annotated).

176. Thus, Suresh teaches the cascading sequences in the first pipeline that interface with one target and the cascading sequences in the second pipeline that interface with two targets, which renders obvious “*the first module plurality of instruction sets being defined for interface with a first group of destination*

applications, the second module plurality of instruction sets being defined for interface with a second group of destination applications,” as claimed.

4. Claim 4

[4.0] *The system of claim 3*

177. See [3.0]-[3.2].

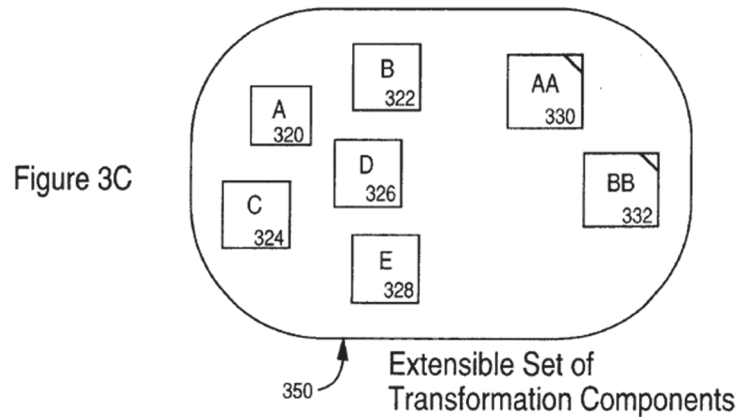
[4.1] *wherein at least one data element of the plurality of data elements is activated by both the first module plurality of instruction sets and the second module plurality of instruction sets.*

178. First, as discussed in [1.1], Suresh describes and suggests an extensible set of ready-made transformation components (“*data elements*”):

In FIG. 3C, an extensible set 350 of transformation components is formed by **transformation components 320, 322, 324, 326, and 328** which were originated from a monolithic transformation application such as M 310, together with transformation components developed by the users (e.g., blocks 330 and 332). **This extensible set 350 of transformation components can be thought of as a pool (i.e., a base of ready-made transformation components from which the users can draw on to build functionally more sophisticated transformations in the data ETL process).**

Suresh, 6:56-65.

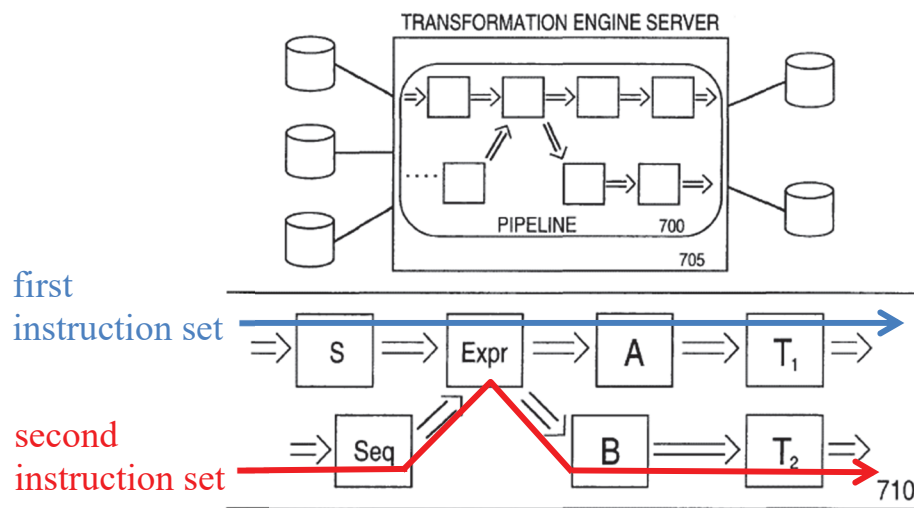
179. The transformation components are also depicted below in Figure 3C:



Suresh, Figure 3C.

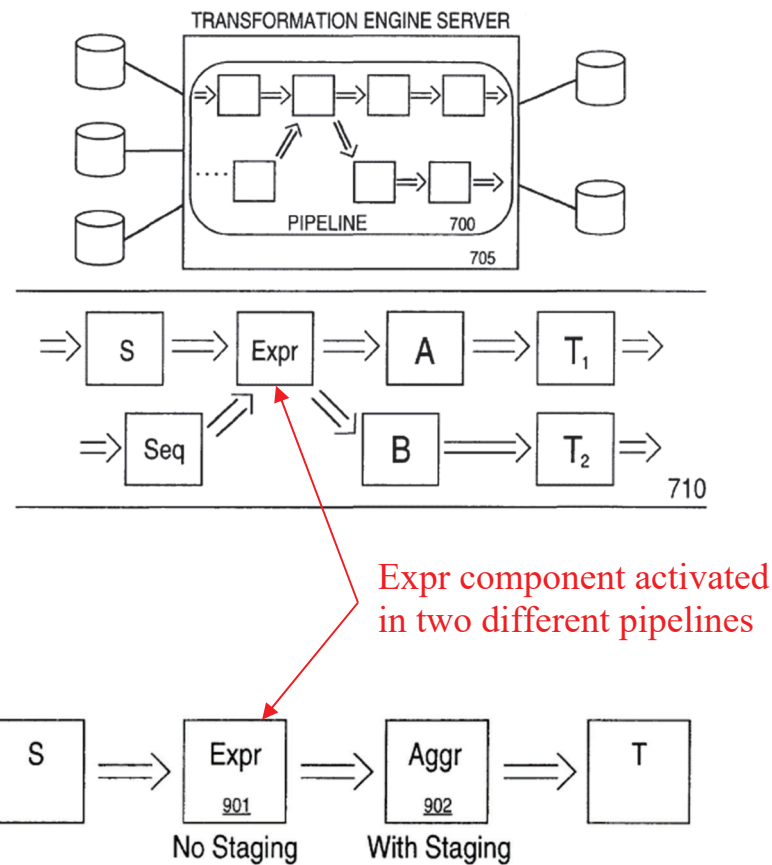
180. Second, Suresh discloses that users can create “pipelined transformations,” such as pipeline 700 and 630 (“*first module* and *second module*”) discussed in [3.1] and [1.2], from the transformation components (“*data elements*”), such as “**transformation components 320, 322, 324, 326, and 328** which were originated from a monolithic transformation application such as M 310, together with transformation components developed by the users (e.g., blocks 330 and 332).” Suresh, 6:56-65; *see also id.*, 7:32-35 (“Different transformations can be added; existing transformations can be removed; designated transformations can be moved to a different portion of the pipeline; additional pipeline stages can be created”).

181. Suresh also describes and suggests that the ready-made transformation components were intended to be used by multiple instruction sets, as exemplified by the “Expr” component in Figure 7:



Suresh, Figure 7 (in part) (annotated).

182. Suresh further describes and suggests that different pipelines can each activate a common transformation component, particularly since the components are being selected from a “pool.” Suresh, 6:56-65 (“transformation components can be thought of as a pool [] i.e., a base of ready-made transformation components from which the users can draw on.”). Suresh also describes example pipelines that each activates a shared common transformation component. *See, e.g.*, Suresh, Figure 7 and Figure 9 (the shared “Expr” component is activated in the different pipelines).



Suresh, Figures 7 and 9 (in part) (annotated)

183. Third, the selected transformation component would be *activated* by both the first module plurality of instruction sets and the second module plurality of instruction sets as discussed in [1.3] and [3.1]. Specifically, Suresh describes and suggests that the transformation component would be *activated* during the ETL processes, which is “[t]he pipelined transformations [that] exist within the transformation engine server 460.” Suresh, 7:27-28.

184. Thus, at least one transformation component of Suresh that (i) is part of a ready-made set of transformation components, (ii) can be included in different

pipelines, and (iii) is activated during the pipelined transformations renders obvious “*at least one data element of the plurality of data elements is activated by both the first module plurality of instruction sets and the second module plurality of instruction sets,*” claimed.

5. Claim 5

[5.0] *The system of claim 1*

185. See [3.0]-[3.3]

[5.1] *wherein the repository further includes configuration data storage for storing configuration data describing data in the source applications, the configuration data storage including a first portion used by the first module and a second portion used by the second module.*

186. First, as discussed in [1.1], Suresh describes and suggests that the repository stores mapping information relating to how data is transformed in a pipeline:

Data relating to operational databases can be entered, updated, and retrieved by clients. A client is used to create a repository, which is used to keep track of session information as well as mapping information relating to how data is to be mapped and transformed from target tables of the operational databases to target tables of data marts.

Suresh, 6:23-29.

187. Second, the mapping information describ[es] data in the source applications because the repository stores mapping information “relating to how data is to be mapped and transformed from target tables of the operational databases to target tables of data marts.” Suresh, 6:23-29; see also id., 7:8-19 (“a pipeline can include a source table 420, an expression transformation 422, an aggregation transformation 424, and a target table 426. The original data, in the form of records, are stored in source table 420.”). Both the information for how data is to be mapped and transformed and the source table that stores the original data teach the configuration data describing data in the source applications. Suresh, 7:66-8:11.

188. The portion of the repository that stores the mapping information, including how the data is to be mapped and transformed and the source table is the configuration data storage. See Suresh, 6:15-29.

189. Third, as discussed in [3.0], Suresh describes and suggests multiple pipelines (“first module” and “second module”), such as pipelines 700 and 630. It would have been obvious to a POSITA that the mapping information of the first pipeline, e.g. pipeline 700 (“first module”) would be stored in a first portion of the configuration data storage and that the mapping information of the second pipeline, e.g., pipeline 630 (“second module”) is stored in a second portion of the configuration data storage. Storing mapping information for different pipelines

allows the transformation engine server to perform “pipelined transformations” by different pipelines in parallel. See Suresh, 8:12-17 (“First of all, a degree of parallelism can be achieved such that one single thread can be dedicated to each of the tasks 630, 640, and 650 inside one process of the computer 660. Secondly, more parallelism can be achieved such that one single process can be dedicated to each of the tasks 630, 640, and 650.”) see also Suresh, 7:32-37 (“multiple processes can run simultaneously within parallel pipeline stages.”)

190. Accordingly, the repository of Suresh that (i) stores mapping information describing data in tables of the source databases and (ii) facilitates executing pipelined transformations in parallel, dedicated threads by storing the mapping information for a first pipeline and a second pipeline, would renders obvious “the repository further includes configuration data storage for storing configuration data describing data in the source applications, the configuration data storage including a first portion used by the first module and a second portion used by the second module,” as claimed.

6. Claim 7

[7.0] *The system of claim 1*

191. See [1.0]-[1.3].

[7.1] *wherein the repository is capable of receiving a plurality of modules.*

192. As discussed in [1.1], Suresh describes and suggests that the

repository contains “transformation components available for use to build functionally more sophisticated transformations.” Suresh, 7:5-8. The users in Suresh would “manually form a new transformation that consists of cascading sequence of transformation components,” such as pipelines 440, 630, 640, 650, and 700. Suresh, 7:9-12, 8:5-9, Figures 4, 6, 7. Suresh discloses that the repository stores the “mapping information relating to how data is to be mapped and transformed from target tables of the operational databases to target tables of data marts,” as the pipeline is executed by the transformation engine server. Suresh, 6:25-29, 8:6-9.

193. It would have been obvious to a POSITA that the user-made pipelines would be stored in the repository once they are created. In this way, the transformation engine server would have easy access to the pipelines when the system administrator requests or schedules the pipeline execution. Further, Suresh describes and suggests that the repository stores mapping information, and the pipelines are the mapping information that describes how the data is transformed using the cascading sequence of the transformation components. *See* Suresh, 6:25-29, 7:9-11. Accordingly, because the pipelines are mapping information, a POSITA would have recognized that the user-made new pipelines would be stored in the repository.

194. Accordingly, Suresh renders obvious “*wherein the repository is*

capable of receiving a plurality of modules,” as claimed.

195. Moreover, Brann describes that the database stores a plurality of translation programs which include the sequence in which data elements are activated. *See* [1.2]. A POSITA would understand that the sequences in which the transformation components to be called in Suresh would be stored in the repository as in Brann.

196. A POSITA would have considered the teachings of Brann and Suresh because they are analogous prior art in the same field of endeavor as the ’178 patent – integrating data among multiple, disparate sources and targets. Suresh, 4:55-57, 6:15-19; Abrams, Abstract; ’178 Patent, Figure 1, 1:5-11

197. Suresh would have benefited from Brann’s teachings. A POSITA would have recognized that the database of Brann could be used to store the subroutines used for the translation, along with the translation instructions, data, and the like. *See*, for example, the rationales set forth for the combination in [1.2].

198. There would have been a reasonable expectation of success in the combination as storing transformation instructions was well-known.

7. Claim 9

[9.0] *The system of claim 1*

199. *See* [1.0]-[1.3].

[9.1] *wherein the active component is capable of loading entire files from a*

source application.

200. Suresh describes and suggests that the transformation engine server (“active component”) can extract data from a data source (“source application”):

[T]hese operational databases provide the data source from which the transformation engine server 240 can extract, transform, then load the data into the databases 250 and 260. Databases 250 and 260 are the data warehouses or data marts that are the targets of the data ETL [Extraction/Transformation/Loading] process.

Suresh, 6:15-20.

201. The data from the source, such as Source A, Source B, or Source C, is loaded using a source transformation component and contains “flat files that provide data for the data mart/data warehouse.” Suresh, 10:39-41. A POSITA would have understood that the flat files for the source transformation component would be loaded from one of the data sources, such as Source A, Source B, or Source C for subsequent transformation by the pipeline. See, e.g., Abrams, 10:56-58 (describing “Source Data” originating from an “ASCII file”). 124. Accordingly, the transformation engine server of Suresh that loads flat files from a data source with a source transformation component renders obvious “wherein the active component is capable of loading entire files from a source application,” as claimed.

8. Claim 10

[10.0] *The system of claim 1*

202. See [1.0]-[1.3].

[10.1] *wherein the repository further includes a library for storing the plurality of data elements.*

203. As discussed in [1.0], Suresh describes and suggests a repository that stores session information and mapping information. See Suresh, 6:23-29. Suresh further describes and suggests that the mapping information includes a pool (“library”) of ready-made transformation components (“data elements”) that are available to build sophisticated transformations:

Again, the extensible set 400 of transformation components is shown here as a repository of transformation components available for use to build functionally more sophisticated transformations.

Suresh, 7:5-8.

This extensible set 350 of transformation components can be thought of as a **pool (i.e., a base of ready-made transformation components from which the users can draw on to build functionally more sophisticated transformations in the data ETL process).**

Suresh, 6:61-65.

204. A POSITA would have recognized that a portion of the repository that

stores a pool of ready-made transformation components is a library.

205. Accordingly, the repository of Suresh that stores a pool of pre-built transformation components renders obvious “wherein the repository further includes a library for storing the plurality of data elements,” as claimed.

9. Claim 23

[23.0] *A system for integrating data among heterogeneous source applications and destination applications, comprising:*

206. See [1.0].

[23.1] *configuration data storage for storing configuration data relating to the source applications;*

207. See [5.1].

[23.2] *temporary data storage for storing temporary data being integrated among the source and the destination applications;*

208. First, as discussed in [1.0], Suresh describes and suggests that the transformation engine server integrates data among the source and destination applications because the transformation engine extracts and transforms data from the source applications and loads the data into the destination applications using pipelined transformations:

In the present embodiment of the present invention, these operational databases provide the data source from which the transformation engine

server 240 can extract, transform, then load the data into the databases 250 and 260. Databases 250 and 260 are the data warehouses or data marts that are the targets of the data ETL [Extraction/Transformation/Loading] process. Suresh, 6:15-20.

Data from any of the source tables 450, 452, and 454 can be transformed by the pipelined transformations within transformation engine server 460 and stored in either and/or both of target tables 470 and 472.

Suresh, 7:28-32.

209. Second, Suresh describes and suggests temporary data storage because it teaches that certain components require staging in RAM and storage disk memory. The RAM and storage disk memory are for storing staged data temporarily as the data is being transformed by one of the transformation components during the pipelined transformation:

The next stage is the aggregation transformation component 940. In contrast to the transformation component expression 920, the aggregation transformation component 940 requires data staging as it transform incoming data fields.

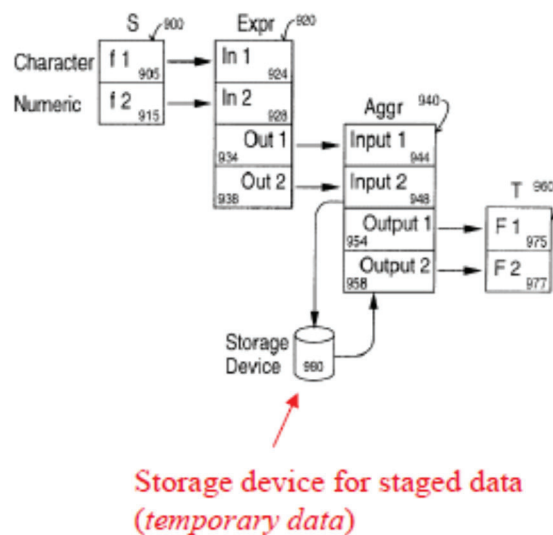
...

Aggregation 940 adds each row entries from field input 2 to a running total that will eventually become output 2 (i.e., the sum of all row entries from

input 2). During the summing process, the quantity of running total needs to be stored (staged) in memory. In other words, data staging becomes necessary because the running total need to be kept track of. Usually, this staging occurs in RAM, then rims out to a storage disk as RAM storage space runs out.

Suresh, 10:13-34.

210. Suresh's Figure 9 also depicts the RAM and storage disk memory ("temporary data storage") that temporarily stores staged data:



211.

Suresh, Figure 9, (in part) (annotated).

212. Suresh further describes and suggests that multiple transformation components have data staging capability:

In the currently preferred embodiment, the source, target, aggregation, rank, and joiner transformations are all staged transformations. The data generated

by these transformations are automatically staged by the software, without human interaction.

Suresh, 10:64-11:1.

213. Thus, the RAM and storage disk memory of Suresh that stores staged data as the data is transformed from the source databases to the target databases using pipelined transformations renders obvious “temporary data storage for storing temporary data being integrated among the source and the destination applications,” as claimed.

[23.3] *a library of data elements, each data element performing a predefined function;*

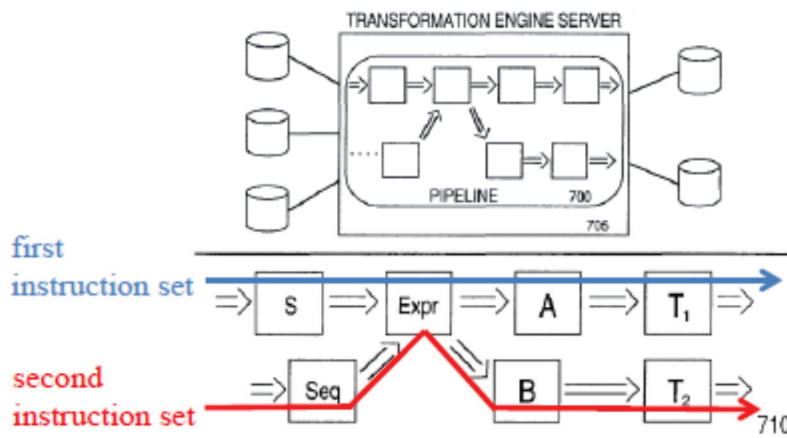
214. First, as discussed in [10.1], Suresh describes and suggests a pool of “ready-made transformation components” which is “a library of data elements.” Suresh, 6:63.

215. Second, Suresh describes and suggests “each data element performing a predefined function” because there are “thirteen different transformation components,” each having a predefined transformation (“predefined function”), including “source, target, expression, aggregation, filter, rank, update strategy, sequence, joiner, lookup, stored procedure, external procedure, and normalizer.” Suresh, 10:35-11:6; see also id., 5:18-21 (“A transformation component is the same as a transformation object, which is a software binary file acting as an

individual unit that possesses built-in autonomy and encapsulates the functionalities of a transformation.”) 138. Accordingly, a pool of thirteen ready-made transformation components of Suresh, each performing a different transformation renders obvious “a library of data elements, each data element performing a predefined function,” as claimed.

[23.4] a first module including a plurality of instruction sets, each instruction set specifying a plurality of data elements to form a chain of data elements; and

216. First, as discussed in [1.2], Suresh describes and suggests a pipeline that is a first module. As also discussed in [1.2], the pipeline includes a plurality of instruction sets because a pipeline may include cascading sequences of transformation components. Suresh’s Figure 7 depicts two example cascade sequences (“plurality of instruction sets”) labeled the first instruction set and the second instruction set, included in the pipeline:



Suresh, Figure 7 (in part) (annotated).

217. Second, as discussed in [1.2], Suresh describes and suggests that each cascading sequence (“instruction set”) specifies transformation components (“a plurality of data elements”). Moreover, because the transformation components are “coupled together (e.g. in a Lego-like fashion),” they form a chain of data elements. Suresh, 7:38-42. For example, the first instruction set depicted in Figure 7 forms a chain of transformation components “ $S \rightarrow \text{Expr} \rightarrow A \rightarrow T1$,” and the second instruction set forms a chain of transformation components “ $\text{Seq} \rightarrow \text{Expr} \rightarrow B \rightarrow T2$.”

218. Thus, a pipeline of Suresh having a plurality of cascading sequences, where each cascading sequence is a series of transformation components coupled together renders obvious “a first module including a plurality of instruction sets, each instruction set specifying a plurality of data elements to form a chain of data elements,” as claimed.

219. To the extent that this claim element is not taught by Suresh, it would be obvious to a POSITA to provide a first module including a plurality of instruction sets, each instruction of each instruction set activating a data element, for example as taught in Brann.

220. Brann teaches a list of subroutines to be invoked and an order in which they should be invoked. Ex. 1003, 7:24-30. The Control Program then invokes the required subroutines in the required order, checking that each

invocation was successful. Id. at 12:1-9. The Control Program extracts its instructions from the database and retains them, in the form of arrays of data structures, in the computer's memory. Ex. 1003, 7:21-35. See VIII. E. 1.2.

221. Thus, Suresh teaching pipelines having a plurality of cascading sequences, where each cascading sequence defines a series of transformation components to be execute in sequence in operation with the transformation engine server to add functionality combined with the teachings of Brann of storing sets of transformation instructions in a database renders obvious “a first module including a plurality of instruction sets, each instruction set specifying a plurality of data elements to form a chain of data elements.”

[23.5] *an active component operably associated with the configuration data storage, the temporary data storage, the library, and the first module,*

222. As discussed in [5.1] and [10.1], Suresh describes and suggests that the configuration data storage and the library are included in the repository because the repository tracks the mapping information relating to how the data is transformed. Suresh, 6:23-29. As discussed in [1.3], the transformation engine server (“active component”) is connected to the repository.

223. As discussed in [23.2], the transformation engine is connected to the RAM and storage disk for storing staged data as the transformation engine executes the pipeline as the pipelined transformation.

224. Because the transformation engine server is (i) connected to the repository (which includes the configuration data storage and the library), (ii) connected to the RAM for storing staged data, and (iii) executes the pipeline as a pipelined transformation, Suresh renders obvious “an active component operably associated with the configuration data storage, the temporary data storage, the library, and the first module,” as claimed.

[23.6] *the active component including a loader for loading data from the source applications to the temporary data storage according to the configuration data relating to the source applications, and*

225. First, Suresh describes and suggests a loader for loading data from the source applications because it teaches a source transformation component (loader) that “extracts from the operational database” data during the ETL process. Suresh, 9:59-62.

226. Second, Suresh describes and suggests that the loader loads the data into the temporary data storage. As discussed in [23.2], Suresh describes and suggests that the data in the source transformation component is “staged” in the RAM and storage disk memory (“temporary data storage.”)

227. Third, Suresh describes and suggests that the data is loaded according to the configuration data relating to the source applications. As discussed in [5.1], Suresh describes and suggests mapping information (“configuration data”) that

describes how data is “mapped and transformed from target tables of the operational databases.” Suresh, 6:23-29; see also id., 7:8-19 (“a pipeline can include a source table 420 ... [and the] original data, in the form of records, are stored in source table 420.”).

228. Fourth, Suresh describes and suggests that the transformation engine server (“active component”) includes the source transformation component (“loader”) because the source transformation component is in the pipeline used by transformation engine server to perform pipelined transformations. Suresh, 7:27-31.

229. Accordingly, the source transformation component that (i) extracts data from the operational database, (ii) then loads the data into the RAM or storage disk memory, (iii) according to the mapping information, and (iv) is included in the pipeline during the pipelined transformation in the transformation engine server renders obvious “the active component including a loader for loading data from the source applications to the temporary data storage according to the configuration data relating to the source applications,” as claimed.

[23.7] *[the active component including ...] an interfacier for populating the destination applications with data from the temporary data storage after the data is processed by the active component,*

230. First, Suresh describes and suggests an interfacier for populating the

destination applications with data ... after the data is processed by the active component because Suresh teaches a target transformation component (“interfacer”) that stores the transformed data into the data marts (“destination applications”) after the transformation engine server (“active component”) transforms the data:

The target transformation maintains database objects or files that receive data from other transformations. These targets then make the data available to data mart users for decision support.

Suresh, 10:41-44.

Data from any of the source tables 450, 452, and 454 can be transformed by the pipelined transformations within transformation engine server 460 and stored in either and/or both of target tables 470 and 472.

Suresh, 7:28-32.

231. Second, Suresh describes and suggests that the target transformation component populates the targets with data from the temporary data storage because the target transformation component stages data in temporary data storage:

In the currently preferred embodiment, the source, target, aggregation, rank, and joiner transformations are all staged transformations. The data generated by these transformations are automatically staged by the software...

Suresh, 10:64-11:1.

Usually, this staging occurs in RAM, then rims out to a storage disk as RAM storage space runs out.

Suresh, 10:30-31.

232. Third, Suresh describes and suggests that the transformation engine server (“activate component”) includes the target transformation component (“interfacer”) because the target transformation component is included in the pipeline that the transformation engine server uses to perform pipelined transformations:

Data from any of the source tables 450, 452, and 454 can be transformed by the pipelined transformations within transformation engine server 460 and stored in either and/or both of target tables 470 and 472.

Suresh, 7:28-32.

233. Accordingly, the target transformation component of Suresh that (i) stages data in the temporary memory, (ii) then loads the transformed data into the data marts, and (iii) is included in the pipeline during the pipelined transformation in the transformation engine server renders obvious “[the active component including ...] an interfacer for populating the destination applications with data from the temporary data storage after the data is processed by the active component,” as claimed.

[23.8] *the active component processing the data by communicating with the first*

module and activating data element chains according to instruction sets of the first module.

234. See [1.3].

10. Claim 24

[24.0] *The system of claim 23*

235. See [23.0]-[23.8].

[24.1] *wherein the plurality of instruction sets of the first module are configured to operate with a particular destination application.*

236. See [2.1].

11. Claim 25

[25.0] *The system of claim 23*

237. See [23.0]-[23.8].

[25.1] *further comprising a second module including a plurality of instruction sets specifying chains of data elements,*

238. See [3.1].

[25.2] *at least one of the data elements specified by the instruction sets of the first module also being specified by the instruction sets of the second module.*

239. See [3.2].

C. Grounds 4 and 5: Claims 8 and 19 are obvious under 35 U.S.C. §103(a) over Suresh in view of Beauchamp or Suresh in view of Brann and Beauchamp

Grounds 4 and 5 are discussed together below. For Ground 5, Brann is used to supplement Suresh for elements 1.0 and 1.2.

1. Summary of Beauchamp

240. Like a configuration interface in Suresh and the '178 patent, Beauchamp discloses “computer tools for designing and implementing integrated enterprise processes on computer systems.” Beauchamp, 1:11-15. One tool in Beauchamp is the “universal client.” Beauchamp, 9:8-12. The universal client provides an “interface for ERP systems, database applications, best-of-breed solutions, desktop applications, legacy systems, and even existing enterprise application integration solutions.” Beauchamp, 9:8-12. Beauchamp further discloses that “[t]he set of standard screens employed through the universal client may be used as building blocks to construct any business process.” Beauchamp, 9:26-28. The “standardized interface screens are linked together in predetermined orders to, in effect, implement on the computer any activity within the class or classes for which the standardized screens are appropriate.” Beauchamp, 4:28-31.

241. Like the '178 Patent, Beauchamp also discloses validation. Beauchamp discloses that validation techniques for validating each component were already well-known. Beauchamp, 2:67-3:2. Beauchamp also describes and

suggests a validator that validates data “based on basic and specialized validation rules.” Beauchamp, 21:32-35.

242. Beauchamp therefore illustrates configuration and validation techniques that were already well-known in the art prior to the earliest alleged priority date of the ’178 patent.

12. Reasons to Combine Suresh and Beauchamp

243. A POSITA would have been motivated to combine Suresh and Beauchamp. Suresh teaches a “transformation engine server 240” that “can extract, transform, then load the data” from source databases into destination databases, Suresh, 6 6-20, and Beauchamp teaches a technique for validating data so “that only proper data entries are received.” Beauchamp, 2:62-64. Specifically, Beauchamp teaches a technique using “code specific validation logic associated with each component,” as well as “basic and specialized validation rules.” Beauchamp, 2:67-3:2 and 21:32-35. Beauchamp also teaches that it is desirable to use a centralized system to interface with “ERP systems [and] database applications.” Beauchamp, 9:8-12.

244. The combination of Suresh and Brann is discussed in ¶¶117-223.

245. First, when considering the teachings of Suresh, a POSITA would have considered the teachings of Beauchamp since they are analogous prior art in the same field of endeavor as the ’178 Patent – integrating data among multiple

applications, including databases. Suresh, 6:15-20, Figure 2; Beauchamp, 4:19-23, 9:8-12, 19:31-34; '178 Patent, Figure 1, 1:5-11.

246. Second, Suresh would have benefited from Beauchamp's teachings in several ways.

a) Storing data in databases of ERP products

247. It would have been obvious to a POSITA to combine Suresh (and Suresh combined with Brann) and Beauchamp to transform and load data into the ERP system that is taught in Beauchamp, and doing so would have been within the POSITA's skillset.

248. As I discussed in the Background section, a dot com boom of the mid and late 1990s gave rise to numerous new companies that developed new software products, including databases. Suresh, for example, explains that "numerous operational databases and many different types of data marts" existed during this time. Suresh, 2:25-32. Due to the sheer number of different systems sold to a business, integrating data among different systems was a well-known problem for business. Beauchamp, 1:42-47. As a result, businesses were looking for different integration solutions for integrating their systems.

249. First, a POSITA considering Suresh would have understood that the ETL process of Suresh could be applied to different types of databases. Beauchamp provides a list of systems known to POSITAs, including an ERP

system, a database application, a database system and a legacy system, amount others, that were being integrated with each other during this time period, and would have been an obvious source of information to a POSITA. Beauchamp, 9:8-12, Figure 11. It was well-known, that an ERP system, like a legacy system, database system, or a database application, included databases. Dahan, Figure 5b; Beauchamp, Figure 11. A POSITA considering Beauchamp would have been motivated to extract, transform, and load data into the databases of the ERP system to integrate data from other databases with the ERP system and make the data accessible to ERP system's applications. In the combination, the ETL process of Suresh would be applied to load data into the database of the ERP system.

250. Second, a POSITA looking to integrate data into various databases, would have looked to transforming and loading data into the ERP system because it was one of many systems with databases that was used by the businesses and needed data integration. *See* Yee, 13:4-20 (Discussing an integration system for integrating data among ERP applications, custom legacy systems, data warehouses, and database management systems.) A POSITA would have recognized the obvious benefits of loading data into the ERP system, including making the data from other databases accessible to the ERP system. There would have been a reasonable expectation of success in the combination because Suresh integrates data among different databases, and Beauchamp's ERP system is just another

system that includes a database.

251. Third, the combination of Suresh and Beauchamp would simply have been combination of prior art elements (integrating Suresh's databases, including those in Beauchamp's ERP system) according to known methods (the ETL technique) to yield predictable results (data integration). The combination of Suresh and Beauchamp would also have been nothing more than applying a known technique (Suresh's ETL process) to a known system (databases Beauchamp's EPR system) in the same way.

b) Validate data during transformation

252. A POSITA considering Suresh would have recognized that data during pipelined transformations should be validated. Suresh, 1:35-42. It was well-known that data in the operational databases is notoriously dirty. Further, data fields that are mandatory for data transformation may not always be populated or be populated with invalid values. Abrams, 7:39, 11:19-22.

253. A POSITA knowing that data in the operational databases is dirty would have looked to techniques for validating the data. This is because performing data transformations on clean or valid data decreases a chance of data corruption and program faults during data transformation. Additionally, data in the source database may be in a format that is different from the target database, e.g.,

month/day/year format instead of day/month/year format, or use a twelve hour clock instead of a twenty-four hour clock, have a different currency format, or telephone number format. Beauchamp, 16:60-62. These formatting issues may cause the data to be transformed improperly. A validation would also be used to make sure that the data is within a valid range, e.g., data that represents a person's salary is a positive number. Yee, 19:5-9. Validation would ensure that the data is in a correct format and reduce the likelihood of errors during data transformation.

254. A POSITA considering Suresh and knowing that data in the operational databases is often dirty, would have looked to other art that describes techniques for validating data. Suresh, 1:35-42. Beauchamp provides those details and would have been obvious source of information to a POSITA. Beauchamp describes validation based on “validation rules” and provides details for using validation rules to correct errors. Beauchamp, 21:27-29, 16:60-65. Beauchamp teaches that the validation rules could be performed in the beginning or at the end of a step or a process. Beauchamp, 16:62-65.

255. First, a POSITA would have been motivated to combine Suresh and Beauchamp to clean and format the dirty data in the operational databases during the pipelined transformations, and doing so would have been within the POSITA's skillset. Using validation rules as taught in Beauchamp would provide a flexible approach for validating different data from different operational databases of

Suresh, correcting invalid, null or of range data values, and formatting the data. Finally, validating data at different steps of the pipelined transformation, e.g., at each transformation component, would ensure that the data remained properly formatted throughout the transformation process.

256. Second, there would have been a reasonable expectation of success in the combination. This is because different data validation techniques were already well-known and applied to validate data. Beauchamp, 2:67-3:2, 16:60-65. Further validating and formatting data when the data is migrated among database systems was known in the art. *See* Abrams, 1:29-37, 5:30-38, 17:1-6; *see* Yee, 19:49-53 (“[t]he system 100 validates every message produced by an adapter 621 or transformer 629 using the validation criteria.”)

257. Third, combining Suresh and Beauchamp would simply have been combining prior art elements (validating Suresh’s data using Beauchamp’s validation techniques) according to known methods (validation rules) to yield predictable results (generating validated, clean, and/or formatted data).

13. Claim 8

[8.0] *The system of claim 1*

258. See [1.0]-[1.3].

[8.1] *wherein the destination applications are included within an ERP product.*

259. Suresh combined with Beauchamp renders this limitation obvious.

260. First, as discussed in [1.0], Suresh describes and suggests a system designed to integrate data between heterogeneous source databases and heterogeneous target databases.

261. Second, Beauchamp teaches that it was desirable for data integration systems to integrate all applications, including ERP systems and other databases:

The universal client may provide **a single business process interface for ERP systems**, database applications, best-of-breed solutions, desktop applications, legacy systems, and even existing enterprise application integration solutions.

Beauchamp, 9:8-12.

The universal clients and process server provide a data driven process framework that provides a structured method for **integrating all of the applications and databases** that a person would involve in a business process.

Beauchamp, 19:31-34.

262. Considering that both Suresh and Beauchamp teach data integration for different applications, including databases, it would have been obvious to a POSITA that the data transformation among heterogeneous systems of Suresh could be used with the example ERP system as disclosed by Beauchamp. *See* Dahan, Figure 5b. In this way, the ERP system in Beauchamp would be able to access and use the data transformed and loaded from other databases applications.

263. Accordingly, the data transformation to destination databases of

Suresh combined with the technique of Beauchamp of providing transformed data to an ERP system render obvious “*wherein the destination applications are included within an ERP product,*” as claimed.

14. Claim 19

[19.0] *The system of claim 1*

264. See [1.0]-[1.3].

[19.1] *wherein each of the plurality of data elements includes related inputs, related outputs, mapping rules, and validation rules.*

265. Suresh and Beauchamp render this limitation obvious.

266. First, Suresh describes and suggests transformation components (“data elements”) that include inputs that receive sets of data inputs and outputs that provide output data sets:

A single transformation can accept one or more sets of inputs from one or more sources and/or prior transformations and can output one or more data sets to one or more subsequent transformations and/or targets.

Suresh, 8:31-34.

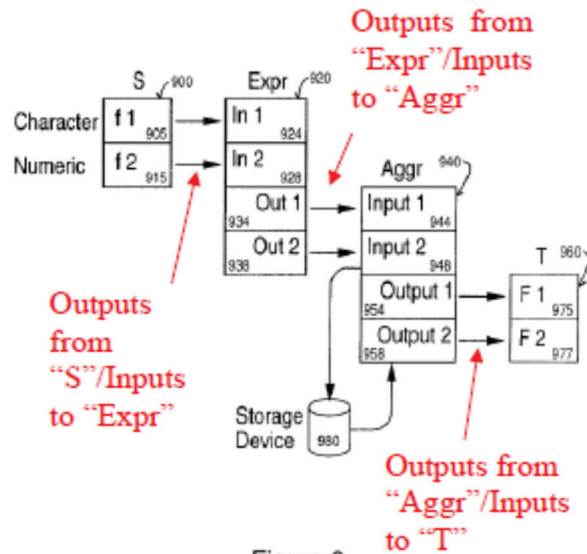


Figure 9

Suresh,

Figure 9 (in part) (annotated).

267. Although not depicted in Figure 9, the source transformation component and target transformation component also include inputs because “source 900 extracts from the source operational database two fields.” Suresh, 9:59-62. Similarly, the target transformation component (“T”) includes outputs because “target transformation maintains database objects or files that receive data from other transformations” and “then make the data available to data mart users.” Suresh, 10:41-44. 232. Suresh also describes and suggests data mapping that occurs at each transformation component (data element):

More specifically, the server establishes the mode of active data movement (e.g., mapping) for each transformation component within each of the pipeline(s) according to the nature/functions of the transformation desired.

Suresh, 9:20-23.

268. The mapping occurs according to mapping rules that are specific to each transformation component. The example mapping rules for Suresh's thirteen different transformation components are replicated below:

[T]here are thirteen different transformation components: source, target, expression, aggregation, filter, rank, update strategy, sequence, joiner, lookup, stored procedure, external procedure, and normalizer. The source transformation contains tables, views, synonyms, or flat files that provide data for the data mart/data warehouse. The target transformation maintains database objects or files that receive data from other transformations. These targets then make the data available to data mart users for decision support. Expression transformations calculate a single result, using values from one or more ports. The aggregation transformation calculates an aggregate value, such as a sum or average, using the entire range of data within a port or within a particular group. Filter transformations filter (selects) records based on a condition the user has set in the expression. The rank transformation filters the top or bottom range of records, based on a condition set by the user.

Suresh, 10:35-52.

For example, a pipeline can include a source table 420, an expression

transformation 422, an aggregation transformation 424, and a target table 426. The original data, in the form of records, are stored in source table 420. The expression transformation 422 performs a calculation based on values within a single record from source table 420 (e.g., based on the price and quantity of a particular item, one can calculate the total purchase price for than [sic] line item in an order). Next, the aggregate transformation 424 is used to perform an aggregate calculation based on all records passed through the transformation (e.g., one can find the total number and average salary of all employees in a particular office using this transformation). The result is then stored as a record in a target table 426.

Suresh, 7:3-25.

269. Second, it would have been obvious to a POSITA that the transformation engine server in Suresh would include rules to validate data at each transformation component. For example, Suresh describes and suggests the aggregation transformation component determines an “output 2=SUM (input 2)” where output 2 is the “sum of all the yearly salaries given out to the employees,” e.g., “the sum of all row entries from input 2.” Suresh, 10:19-26. It would have been obvious to a POSITA to validate that the data in input 2 and output 2 would be mapped to a numeric field (and not character fields) when a user manually selects components for inclusion into a pipeline or that the data in the input 2 and

output 2 is not a negative number. Suresh, 5:36-41. 235. Third, in addition to Suresh's teaching of validating output, Beauchamp shows that using validation rules was known:

When the user clicks on one of the navigation options, an action event triggers an action manager 258 to prepare the client data to return to the server. A validator 260 first validates all of the data currently maintained in the value monitor 256 based on basic and specialized validation rules that it receives from the XML resolver 252. Once all data validations have passed, the action manager 258 passes along all changed data to an XML generator 262 which formats the XML for the trip back to the process server.

Beauchamp, 21:29-38.

270. Beauchamp further teaches that validation can be performed "at the beginning or end of a step, process." Beauchamp, 16:62-65.

271. In a combination, it would have been obvious to a POSITA to use Beauchamp's validation rules to validate the data in Suresh. Validating data would ensure the data is valid and remains valid throughout the transformation processes. Validation would be advantageous because it would clean empty, invalid, null, out-of-range values, and format the data. See Beauchamp, 2:64-67. Validation would also reduce the chances of a memory being overwritten by operating on incorrect or invalid data and correct the dirty data that is often found in the source databases.

Finally, using validation rules while integrating data was well-known in the art.

Yee, 19:49-53, 30:60-63.

272. Thus, a combination of Suresh and Beauchamp teaches data elements that include inputs, outputs, mapping rules and validation rules, which renders obvious “each of the plurality of data elements includes related inputs, related outputs, mapping rules, and validation rules,” as claimed.

IX. CONCLUSION

This declaration and my opinions herein are made to the best of my knowledge and understanding, and based on the material available to me, at the time of signing this declaration. I declare that all statements made herein on my own knowledge are true and that all statements made on information and belief are believed to be true, and further, that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 or Title 18 of the United States Code.

Date: 2/21/24Henry Houh.

Henry Houh