

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

GOOGLE LLC,
Petitioner,

v.

DIALECT, LLC,
Patent Owner.

Case IPR2024-00752
Patent 8,849,652

PATENT OWNER'S RESPONSE

TABLE OF CONTENTS

I.	INTRODUCTION	1
II.	THE ’652 PATENT DISCLOSES AND CLAIMS PROCESSING NATURAL LANGUAGE UTTERANCES, AND SYNCHRONIZING A FIRST CONTEXT STACK IN ONE DEVICE WITH A SECOND CONTEXT STACK IN ANOTHER DEVICE AS PART OF THE PROCESSING	4
III.	THE BOARD NEED NOT REACH ANY CLAIM CONSTRUCTION ISSUES	12
IV.	PETITIONER’S CHALLENGES FAIL ON THE MERITS	14
A.	Overview of Coffman.....	17
i.	Coffman’s Non-Proprietary Speech Engine and Application Program Interfaces (APIs) Receive Speech Inputs from Con conversationally-Aware Applications But Do Not Select or Invoke Conversationally-Aware Applications.....	19
ii.	Coffman’s Context Stacks are Unique to Each Application, and Searching an Application’s Context Stack Does Not Result in Identifying Another Application to Process a User’s Command or Request	28
iii.	Coffman Also Does Not Synchronize Context Stacks or Context Sets.....	29
B.	Petitioner’s Ground 1 Fails to Satisfy Synchronizing a First Context Stack (or Context Set) with a Second Context Stack (or Context Set).....	32
C.	Petitioner’s Ground 1 Also Fails to Satisfy the Identification of Domain Agents Configured to Process a Command or Request of Dependent Claims 8, 18, and 24	37
V.	CONCLUSION.....	47

TABLE OF AUTHORITIES

Other Authorities

Consolidated Trial Practice Guide (Nov. 2019) 36

Rules

37 C.F.R. § 42.65(a).....36

37 C.F.R. § 42.120 1

PATENT OWNER'S EXHIBIT LIST

Exhibit No.	Description
2001	Patent Owner's Power of Attorney
2002	U.S. Patent 7,693,720 ("the '720 Patent")
2003	U.S. Patent 8,015,006 ("the '006 Patent")
2004	U.S. Patent 8,195,468 ("the '468 Patent")
2005	U.S. Patent 9,263,039 ("the '039 Patent")
2006	U.S. Patent 9,495,957 ("the '957 Patent")
2007	Order (Construing Claim Terms), <i>Dialect, LLC v. Amazon.com Inc.</i> , Civ. No. 1:23cv581, Dkt. 213 (E.D. Va. Apr. 29, 2024)
2008	U.S. Patent Application Publication No. 2007/0043574 ("Coffman Counterpart")
2009	Cross-reference between Petitioner's Citations to Coffman and Coffman Counterpart
2010	Reserved
2011	Deposition Transcript of Christopher Schmandt, taken January 29, 2025
2012	<i>Google LLC v. Dialect, LLC</i> , IPR2024-00746 Institution Decision, Paper 11 (P.T.A.B. Oct. 15, 2024)
2013	Declaration of Benedict Occhiogrosso in Support of Patent Owner's Response

Pursuant to 37 C.F.R. § 42.120, Patent Owner Dialect, LLC (“Dialect” or “Patent Owner”) files this response to the Petition filed by Google LLC (“Petitioner”), setting forth reasons why the Petition for *inter partes* review (“IPR”) fails to establish the unpatentability of claims 1-27 of U.S. Patent No. 8,849,652 (the “’652 patent”). This Response is supported by the expert declaration of Benedict Occhiogrosso, submitted as Ex. 2013 (hereinafter, “Occhiogrosso”).

I. INTRODUCTION

The ’652 patent is directed to enabling natural language human-machine interactions with speech inputs. The invention uses context stacks and context stack entries in the course of performing its interpretation of the speech inputs. Central to the invention claimed in the ’652 patent, when context stacks (or sets of contexts) are present in different devices, the ’652 patent synchronizes a first context stack or context set in a first device with a second context stack or context set in a second device.

The Petition relies upon a single reference—Coffman—to disclose the claimed “synchronizing.” But Petitioner fails to establish that Coffman synchronizes a “context stack” (or “context set”) in one device with another “context stack” (or “context set”) in another device. Petitioner’s citations to Coffman do not mention “synchronizing” and, instead, just indicate that a master

dialog manager in Coffman may instruct a slave dialog manager *to add or update* context in a context stack. But even adding or updating one stack in view of a second is not sufficient to establish that two stacks are synchronized, as each stack would possess a number of varying attributes including the contents, size, and order of the stacks.

Indeed, during deposition, Petitioner's expert Mr. Schmandt was asked about two "stacks [with] the same content but in different orders within their stacks," and whether "a POSITA [would] have viewed those stacks as identical?" Mr. Schmandt's response was unequivocal, "No, because order is a – order is an attribute of stack." Ex. 2011, 13:10-17. Mr. Schmandt's testimony confirms that for two stacks to be synchronized, it is not enough that they contain the same entries; they also must contain those entries in an identical order.

Patent Owner's expert also agrees with this opinion (Occhiogrosso, ¶73), and explains that a POSITA would not interpret two stacks containing different context entries, or even equivalent context entries but in different orders, as being synchronized. As a result, Coffman's disclosure that a master dialog manager may instruct a slave dialog manager *to add or update* context in a context stack is not sufficient to conclude that these stacks are "synchronized."

Indeed, Petitioner provides no other evidence or reasoning that would establish the stacks or sets themselves in different devices would be synchronized

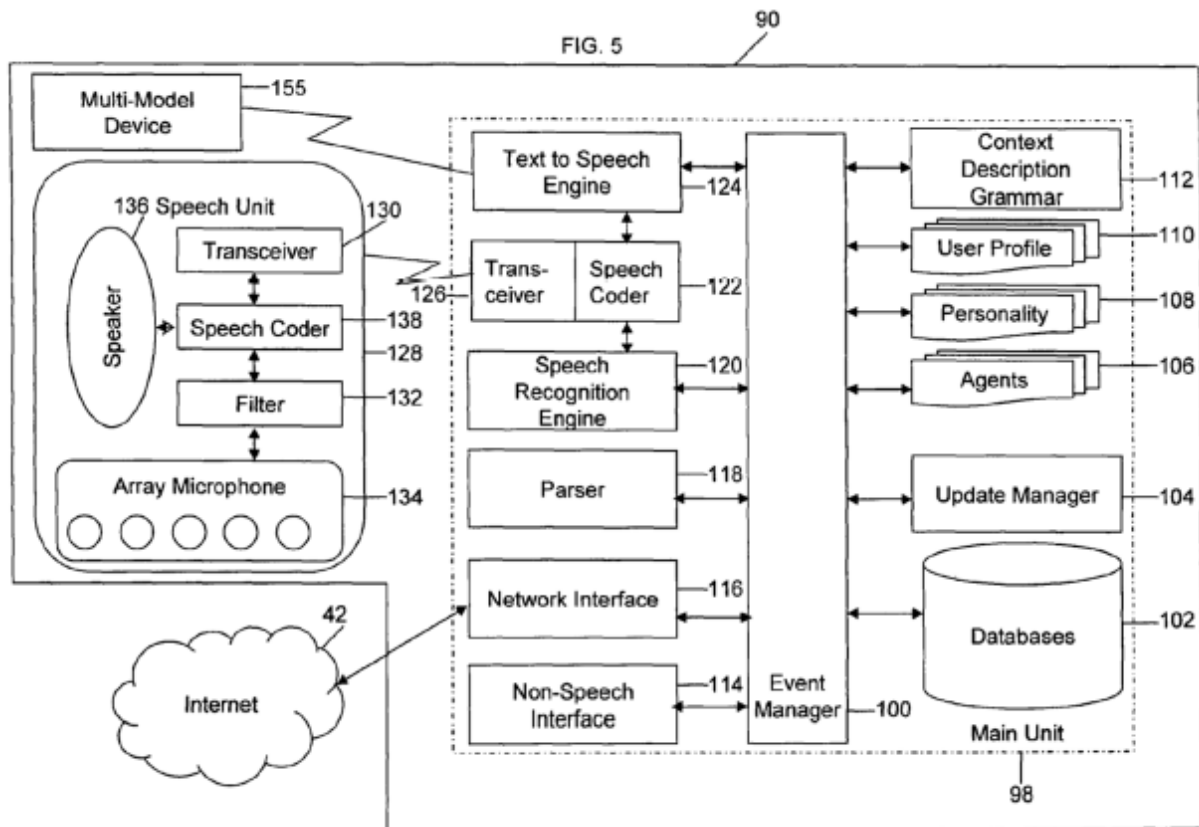
as required by the claims of the '652 patent, which happens to be the precise reason for the Examiner's allowance of the claims. Ex. 1012, 447 ("the limitation ... is also more narrow in that not only is the context synchronized, but context stacks/sets of distinct devices are synchronized.") (emphasis in original). In fact, such a result—synchronized context stacks in different devices—is inconsistent with Coffman, which states that "different devices, CVMs or different applications *will have their own ... context stack 217, 217a.*" Coffman, 27:1-2 (emphasis added).

Additionally, Petitioner's challenges of at least dependent claims 8, 18, and 24 fail. As the Board concluded in IPR2024-00746, -00747, -00748, -00749, and -00751, Petitioner's interpretation of Coffman's relationship to the domain agents claimed in the patents-at-issue in those cases was deficient, and those petitions were denied. Here again, Petitioner's challenges of certain dependent claims related to the selection of a domain agent to process a user's utterance suffers from the same defects. For the same reasons that those other IPRs were denied institution, here Petitioner fails to establish the unpatentability of at least dependent claims 8, 18, and 24, along with their dependent claims 9-11 and 25-27.

For these reasons, Petitioner's challenges fail.

II. THE '652 PATENT DISCLOSES AND CLAIMS PROCESSING NATURAL LANGUAGE UTTERANCES, AND SYCHRONIZING A FIRST CONTEXT STACK IN ONE DEVICE WITH A SECOND CONTEXT STACK IN ANOTHER DEVICE AS PART OF THE PROCESSING

The '652 patent is titled, "Mobile Systems and Methods of Supporting Natural Language Human-Machine Interactions." FIG. 5 of the '652 patent, reproduced below, is an overall diagrammatic view of an embodiment of the '652 patent:



'652 Pat., FIG. 5, *see also id.*, 15:38-40. The system includes, among other things, a main unit 98, speech unit 128, speech recognition engine 120, context description

grammar module 112, parser 118, agents 106, multi-modal device 155 and non-speech interface 114. '652 Pat., 23:38-39, 25:60-65.

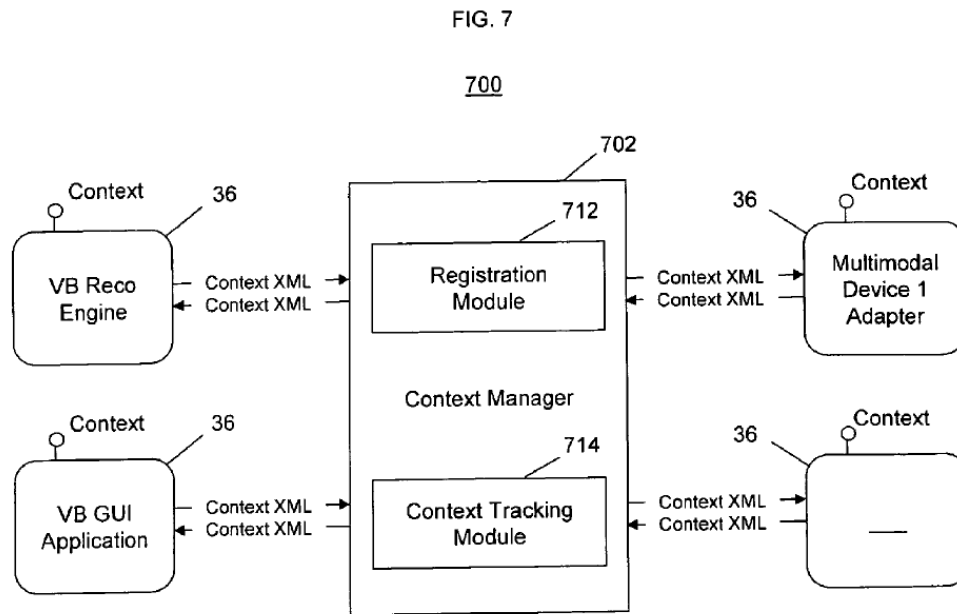
Speech unit 128 includes a microphone to receive a spoken utterance from a user. '652 Pat., 25:50-52. After initial processing, the speech is passed to speech recognition engine 120 for processing. '652 Pat., 26:18-20. Then, "[a]ny recognized information may be processed by the parser 118, which transforms information into complete algorithms and questions using data supplied by knowledge agents." '652 Pat., 26:20-23.

More particularly, after speech recognition, parser 118 "may determine a context for an utterance by applying prior probabilities or fuzzy possibilities to keyword matching, user profile 110, dialog history, and context stack contents." '652 Pat., 34:2-5. "For example, a question with the keywords 'temperature' implies a context value of weather for the question." '652 Pat., 34:7-8. "Once the context for the question or command has been determined, the parser 118 can invoke the correct agent 156, 150." '652 Pat., 34:34-36.

To determine the context, the '652 patent discloses the use of a context stack. The context stack "includes an ordered list of command contexts, to enable follow-up requests." '652 Pat., 20:10-12; *see also* '652 Pat., 36:13-14 ("Part of the dialog history is maintained in a context stack.")).

The system of the '652 patent may be incorporated in a plurality of devices. Data and agents may be stored and synchronized in mobile structures 10, mobile devices 36 and/or fixed systems 44 having one or more main speech-processing units 98. '652 Pat., 17:15-32.

FIG. 7 of the '652 Patent depicts multiple mobile devices 36:



Each mobile device 36 may track context. '652 Pat., 19:32-33. A context manager 702 may be centrally maintained to allow input to and output from multiple mobile devices 36. '652 Pat., 19:34-36.

Generally, the synchronization between different systems in the '652 patent can occur on the wide area wireless network 38, the data network 42, through the handheld interface 20, or other local data connections. '652 Pat., 17:18-21. The synchronization process attempts to determine which version of a data element or

an agent are the newest or most up-to-date and propagates that element. '652 Pat., 17:25-28. Thus, synchronization can be an incremental change process. *Id.*, 17:28-29. In some cases, a complete replacement of a database, a portion of the database or of one or more agents may be performed rather than performing a series of incremental updates. *Id.*, 17:29-32. The synchronization may be performed automatically when any two or more of the computers are connected to these networks. *Id.*, 17:21-23. Alternatively, the synchronization may be performed on demand under user control. *Id.*, 17:24-25.

The synchronization of contexts in the '652 patent, whether that synchronization covers the entire stack or a set of stack entries, preserves the ordered nature of the context stack or the set of context entries (or "contexts") in the context stack. For example, as stated in the '652 patent, a context stack can include "an ordered list of command contexts," '652 Pat., 20:10-12, and a "set" of contexts can be stored in the context stack. '652 Pat., 20:10-12. Additionally, the order of entries in the context stack is meaningful because the order of the entries may be updated based on rank scores while determining a user's command or request. '652 Pat., 39:23-27 (claim 5). Accordingly, as Mr. Occhiogrosso explains, a POSITA would understand that a context stack and a context set each provide an ordered list of contexts, and that by synchronizing two context stacks or

sets, the ordered list of a target context stack or set would be synchronized with the ordered list of a source context stack or set. Occhiogrosso, ¶39.

In FIG. 7, and with regard to context stack synchronization, each mobile device 36 “may communicate with the context manager 702 to register via registration module 712, wherein registration may indicate events that the mobile devices 36 may subscribe to.” ’652 Pat., 19:33-39. The context manager 702 may receive input in Context XML form, for example. ’652 Pat., 19:39-41.

Other registered mobile devices 36 may be informed of context changes through context tracking module 714, thereby synchronizing the context across the registered mobile devices 36. ’652 Pat., 19:41-48; *see also id.*, 4:27-32.

“The knowledge agents may then process the commands or questions.” ’652 Pat., 26:27-31. Non-speech interface 114 and multi-modal devices 155 may be used as a substitute or complement to the speech interface. ’652 Pat., 26:51-53.

Consistent with the ’652 patent’s disclosure, independent claims 1 and 14 of the ’652 patent require synchronization of a context stack, and claim 20 requires synchronization of the context set. Claim 1 recites:

1. A system for processing natural language utterances where recognized words of the natural language utterances alone are insufficient to completely determine one or more commands or requests, the system comprising:

one or more physical processors programmed with one or more computer program instructions which, when executed, cause the one or more physical processors to:

generate a first context stack associated with a first device, the first context stack comprising context information that corresponds to a plurality of prior utterances;

synchronize the first context stack with a second context stack associated with a second device such that the context information of the first context stack is updated based on related context information of the second context stack;

receive a natural language utterance associated with a command or request;

determine one or more words of the natural language utterance by performing speech recognition on the natural language utterance; and

determine the command or request based on the one or more words and the updated context information.

'652 Pat., Claim 1 (emphasis added).

Claim 14 recites:

14. A computer-implemented method of processing natural language utterances where recognized words of the natural language utterances alone are insufficient to completely determine one or more commands or requests, the method being implemented by a computer system that includes one or more physical processors executing one or more computer program instructions

which, when executed, perform the method, the method comprising:

generating, by the one or more physical processors, a first context stack associated with a first device, the first context stack comprising context information that corresponds to a plurality of prior utterances;

synchronizing the first context stack with a second context stack

associated with a second device such that the context information of the first context stack is updated based on related context information of the second context stack;

receiving, at the one or more physical processors, a natural language utterance associated with a command or request;

determining, by the one or more physical processors, one or more words of the natural language utterance by performing speech recognition on the natural language utterance; and

determining, by the one or more physical processors, the command or request based on the one or more words and the updated context information.

'652 Pat., Claim 14 (emphasis added).

Claim 20 recites:

20. A computer-implemented method of processing natural language utterances where recognized words of the natural language utterances alone are insufficient to completely determine one or more commands or requests, the method being implemented by a computer system that includes one or more physical processors executing one or more computer program instructions

which, when executed, perform the method, the method comprising:

generating, by the one or more physical processors, a first context set associated with a first device, the first context set comprising context information that corresponds to a plurality of prior utterances;

synchronizing the first context set with a second context set

associated with a second device such that the context information of the first context set is updated based on related context information of the second context set;

receiving, at the one or more physical processors, a natural language utterance associated with a command or request;

determining, by the one or more physical processors, one or more words of the natural language utterance by performing speech recognition on the natural language utterance; and

determining, by the one or more physical processors, the command or request based on the one or more words and the updated context information.

'652 Pat., Claim 20 (emphasis added).

The recitations directed to synchronizing context stacks or context sets in different devices were first presented in dependent claim 63 of the application that led to the '652 patent. Ex. 1012, 127. During examination, the Examiner issued an Office Action finding that the synchronizing requirements presented allowable subject matter, reasoning "the limitation ... is also more narrow in that not only is the context synchronized, but context stacks of distinct devices are synchronized."

Ex. 1012, 170-71 (emphasis in original). In response to the Examiner's Office Action, Patent Owner presented the requirements for synchronization of context stacks or sets in the independent claims (Ex. 1012, 186 (claim 50), 191 (claim 64), 194 (claim 70) & 198), and the Examiner passed the application to allowance (Ex. 1012, 446), again reasoning that "the limitation ... is also more narrow in that not only is the context synchronized, but context stacks/sets of distinct devices are synchronized." *Id.*, 447 (emphasis in original). As acknowledged in the Petition, the Examiner identified Coffman's U.S. patent counterpart as a cited reference and considered it and Kennewick prior to allowing the '652 patent. Pet., 14-15; *see also* Ex. 1012, 175, 216, 224.

III. THE BOARD NEED NOT REACH ANY CLAIM CONSTRUCTION ISSUES

The Petitioner identifies no terms as requiring constructions. The United States District Court for the Eastern District of Virginia has issued constructions for the following terms in other patents assigned to Patent Owner (Ex. 2007):¹

¹ In this table, the '652 patent is the parent of U.S. Patent 9,495,957 ("the '957 Patent") (Ex. 2006), and U.S. Patent 8,195,468 ("the '468 Patent") (Ex. 2004) is related to the '652 patent through a continuation. Ex. 2006, 1-2; '652 Patent, 1. U.S. Patent 7,693,720 ("the '720 Patent") (Ex. 2002), U.S. Patent 8,015,006 ("the '006 Patent") (Ex. 2003), and U.S. Patent 9,263,039 ("the '039 Patent") (Ex. 2005) are each assigned to Patent Owner and share at least one inventor in common with the '652 patent.

Term	Patent(s)	Claim(s)	Construction
context	'720 Patent	1, 31	"the subject matter area to which a particular user input is directed and which is used to determine the meaning of the user input"
	'006 Patent	1, 2, 4, 5, 10	
	'468 Patent	19, 28	
	'039 Patent	17, 18	
context stack	'006 Patent	2	"a data structure that stores or references contexts in a manner that can be ordered"
	'468 Patent	19, 28	
	'039 Patent	17	
	'957 Patent	1, 3, 4, 7	
entries in a context stack	'468 Patent	19	"elements of a context stack that store or reference contexts"
context entries	'957 Patent	1, 3, 4, 5, 7	"elements of a context stack that store or reference contexts"
grammar	'720 Patent	1, 31	"a set of rules governing how inputs and requests to a domain agent should be structured"
	'006 Patent	1, 5, 10	
context description grammar	'039 Patent	13	"a data structure containing entries constituting or referencing sets of rules, wherein each of those sets describes the structure of natural language in a particular context"
domain agent	'720 Patent	1, 31	"executables associated with a specific domain that receive, process and respond to user questions, queries and commands"
	'006 Patent	1, 2, 5, 10	
	'468 Patent	19, 30, 32	"software with domain-specific behavior and information"
	'039 Patent	13, 14	

procedures sensitive to the determined context	'006 Patent	5	"procedures that differ based on which context has been determined"
Parser	'720 Patent	1, 35	"software that analyzes a string of words"
	'006 Patent	1, 5, 10	

For the substantive reasons presented in Section IV below, Petitioner's grounds include fundamental flaws and fail to establish the disclosure of every claimed element in each independent claim. These findings do not require the Board to conduct any construction of the above-identified terms.

IV. PETITIONER'S CHALLENGES FAIL ON THE MERITS

Petitioner asserts three Grounds in its Petition. Ground 1 is asserted against each independent claim, and contends that claims 1-9, 14-18, and 20-26 are obvious in view of Coffman alone.² Grounds 2 and 3 are not used to shore up the deficiencies of Ground 1 and, instead, are asserted against certain dependent claims. Ground 2 contends claims 10-13, 19, and 27 are obvious in view of Coffman and Kennewick³. Ground 3 contends that dependent claims 13 and 19 are obvious in view of Coffman, Kennewick, and Lee⁴.

² Ex. 1003, International Patent Application Publication No. WO 00/20962 ("Coffman").

³ Ex. 1004, U.S. Patent App. Pub. No. 2004/0044516 ("Kennewick").

⁴ Ex. 1006, U.S. Patent App. Pub. No. 2002/0087315 ("Lee").

As noted above, Coffman's disclosures were known to the Examiner through Coffman's U.S. patent publications, and they were specifically identified by the Examiner. *See* Ex. 1012, 175, 216, 224; *see also* Pet., 14-15. The Petition tries to create some mystery with the Examiner's allowance by alleging that "neither of Coffman's related U.S. patent publications ... were discussed during prosecution." Pet., 14. But there is no intrigue; the Examiner correctly allowed the claims over Coffman's counterparts. The distinction expressly emphasized in the Examiner's reason for allowance—that even synchronizing a context entry does not show synchronization of context stacks or synchronization of context sets—is a distinction between Coffman and the challenged claims as well. As discussed below, Coffman may add context to a context stack or update context in a context stack, but that does not establish that Coffman discloses ***synchronization*** of a context stack or context set, which also requires synchronization of order of the context stack or context set. By only disclosing that individual entries may be added or updated, Coffman provides no mechanism for synchronizing any other attribute of the context stack or context set, including the attribute of context order. Accordingly, Coffman fails to satisfy the synchronization requirements recited in the independent claims.

Additionally, without disclaiming the independent patentability of any other dependent claim, Petitioner's challenges of dependent claims 8, 18, and 24 also

fail. As the Board concluded in IPR2024-00746, -00747, -00748, -00749, and -00751, Petitioner's interpretation of Coffman's relationship to the domain agents claimed in the patents-at-issue in those cases was deficient, and those petitions were denied. Specifically, the Board agreed in those prior IPRs that Petitioner failed to correlate Coffman's conversationally-aware applications with the claimed "domain agent." In the '652 patent, claims 8, 18, and 24 recite features related to the identification of a domain agent configured to process a user's request or command, and Petitioner's challenges of these claims suffers from the same defects as in the prior-recited IPRs. Specifically, Petitioner again relies on Coffman's conversationally-aware applications as disclosing the claimed "one or more domain agents" of these dependent claims, and does so on a substantially similar basis as in the prior-recited IPRs. For the same reasons that the above-identified IPRs were denied institution, here Petitioner fails to establish the unpatentability of at least dependent claims 8, 18, and 24, along with dependent claims 9-11 and 25-27. *Occhiogrosso*, ¶¶92-94.

Thus, Ground 1 is facially deficient in establishing the unpatentability of any challenged independent claim, and Grounds 2 and 3 do not remedy the deficiencies of Ground 1. For the reasons discussed below, the Petition fails to establish unpatentability of any challenged claim.

A. Overview of Coffman

In FIG. 6, reproduced below, Coffman discloses a conversational input/output interface (Coffman, FIG. 6 & 37:14-29):

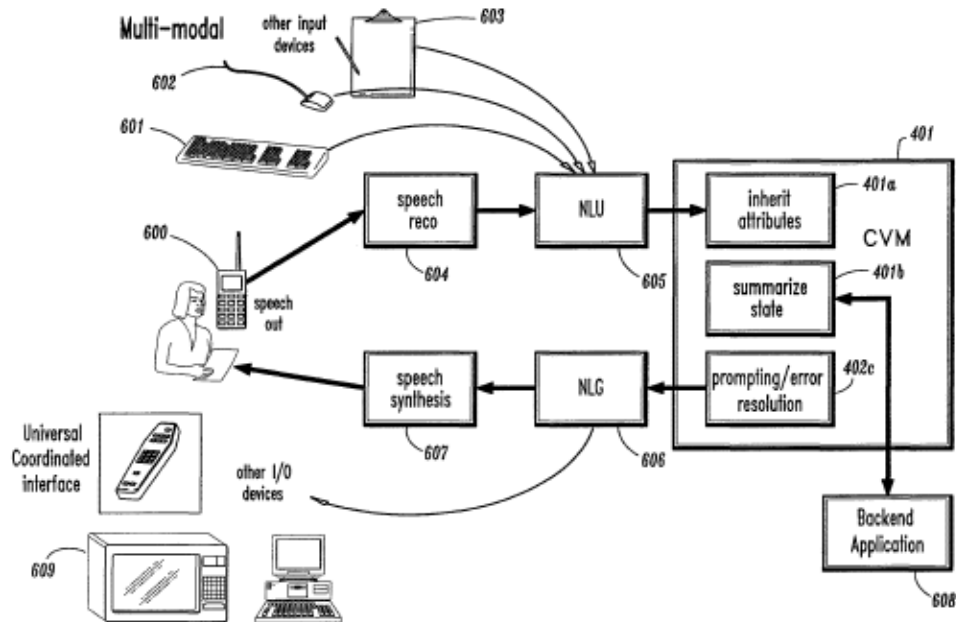
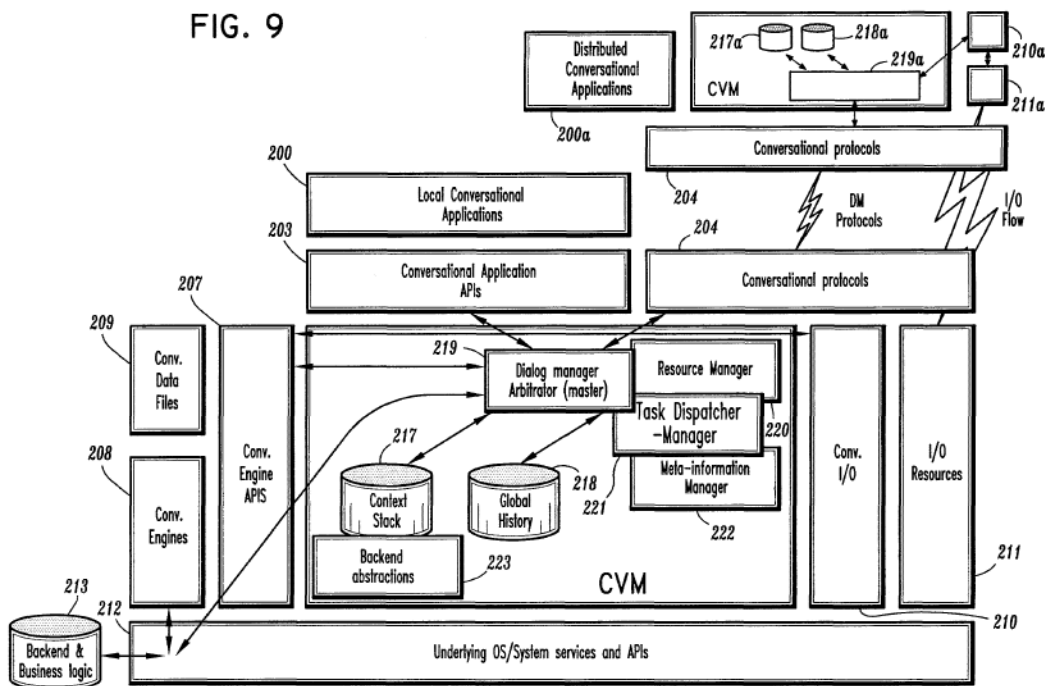


FIG. 6

The conversational input interface according to Coffman can process multi-modal inputs, such as files/streams/resources, speech via a phone 600, keyboard 601, pointing devices 602, and handwriting devices 603. The modalities are caught and transferred to a dialog manager that causes the inputs to be stored in the context stack. Spoken input from a speech client is subject to a speech recognition process 604 and other input are subject to NLU [Natural Language Understanding] processing 605.

As illustrated in FIG. 9, reproduced below, the conversational protocols of Coffman include protocols for information exchange between dialog managers (DMs) of networked devices (Coffman, 26:28-30):



When applications are networked (distributed applications 200a), dialog management protocols are used for exchanging information to determine which dialog manager (219 or 219a) will execute a given function. Coffman, 26:30-27:1. As Coffman explains, typically, “different devices, CVMs [Conversational Virtual Machines] or different applications *will have their own* dialog manager, *context stack 217, 217a* and global history 218, 218a.” Coffman, 27:1-2 (emphasis added).

Through the dialog manager DM protocols, the different dialog managers will negotiate a topology. Coffman, 27:2-5. An active master dialog manager (illustrated as dialog manager 219 in FIG. 9) will be responsible for managing the flow of I/O to the different managers to decide the active dialog and appropriately execute a query and update the context and history. Coffman, 27:5-7.

i. Coffman's Non-Proprietary Speech Engine and Application Program Interfaces (APIs) Receive Speech Inputs from Conversationally-Aware Applications But Do Not Select or Invoke Conversationally-Aware Applications

Coffman was seeking to provide a backend system that would more easily allow a developer to program a voice-based interactive telephone application to converse with a user, without having to build an entire backend speech engine or API. Coffman notes that “voice command and control (voice C&C) UIs [user interfaces] are emerging everywhere computers are used.” Coffman, 3:1-2. “For instance, telephone companies, call centers and IVR have implemented speech interfaces to automate certain tasks, reduce their operator requirements and operating costs and speed-up call processing.” Coffman, 3:5-7. However, Coffman also notes that interactive voice response (“IVR”) “application developers offer their own proprietary speech engines and APIs (application program interface),” and that “proprietary applications are typically not portable from vendor to vendor.” Coffman, 3:7-8.

Coffman also states that “speech interfaces for GUI based OSs have been implemented using commercially available continuous speech recognition applications for dictation and command and control.” Coffman, 3:12-14. Coffman describes these “applications” as “add-ons to the GUI based OSs in the sense that such applications allow for the replacement of keyboard and mouse and allows a user to change the focus, launch new tasks, and give voice commands to the task in focus.” Coffman, 3:14-16.

As explained, Coffman's purpose is to provide a “conversational virtual machine” with a speech engine and application program interfaces (APIs) usable by application developers “for implementing conversational computing” with conventional applications and conversationally-aware applications. Coffman, 1:7-17. The point of Coffman, therefore, is to provide a “universal conversational user interface.” Coffman, 3:30.

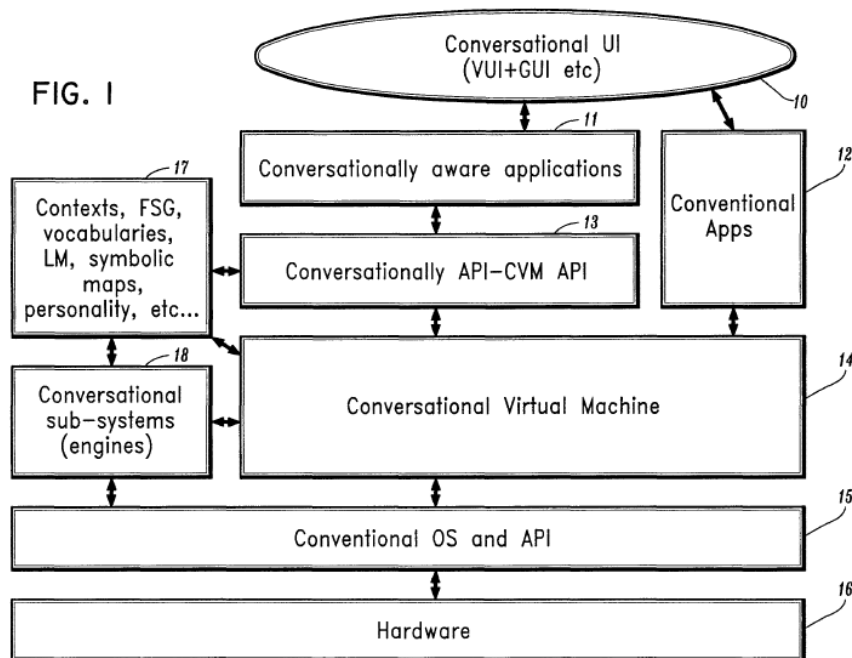
Coffman's Conversational Virtual Machine, or “CVM,” “utilizes conversational subsystems (which may be local or distributed) including speech recognition, speaker recognition, text-to-speech, natural language understanding and natural dialog generation engines to understand and generate dialog between ... user and machine.” Coffman, 5:1-4. Coffman refers to these aspects of the CVM as a “subsystem,” and notes that the “subsystem [is] accessed through the CVM.” Coffman, 5:4-5. As an aid to software developers, Coffman explains that

the “development environment offered by the CVM ... allows a developer to build, simulate and debug conversational aware application [*sic*] for CVM.” Coffman, 18:16-18. An application developer does not need to build a speech engine or APIs, and instead accesses the CVM through the system's established APIs. The system's speech “engines are hidden to the application through the conversational application APIs.” Coffman, 5:5-6.

The relationship of Coffman's CVM to conventional applications, conversationally-aware applications, and the APIs is illustrated in Coffman's FIG. 1.

1. Coffman's FIG. 1 is described as a “block diagram of a conversational computing system according to an embodiment of the present invention.”

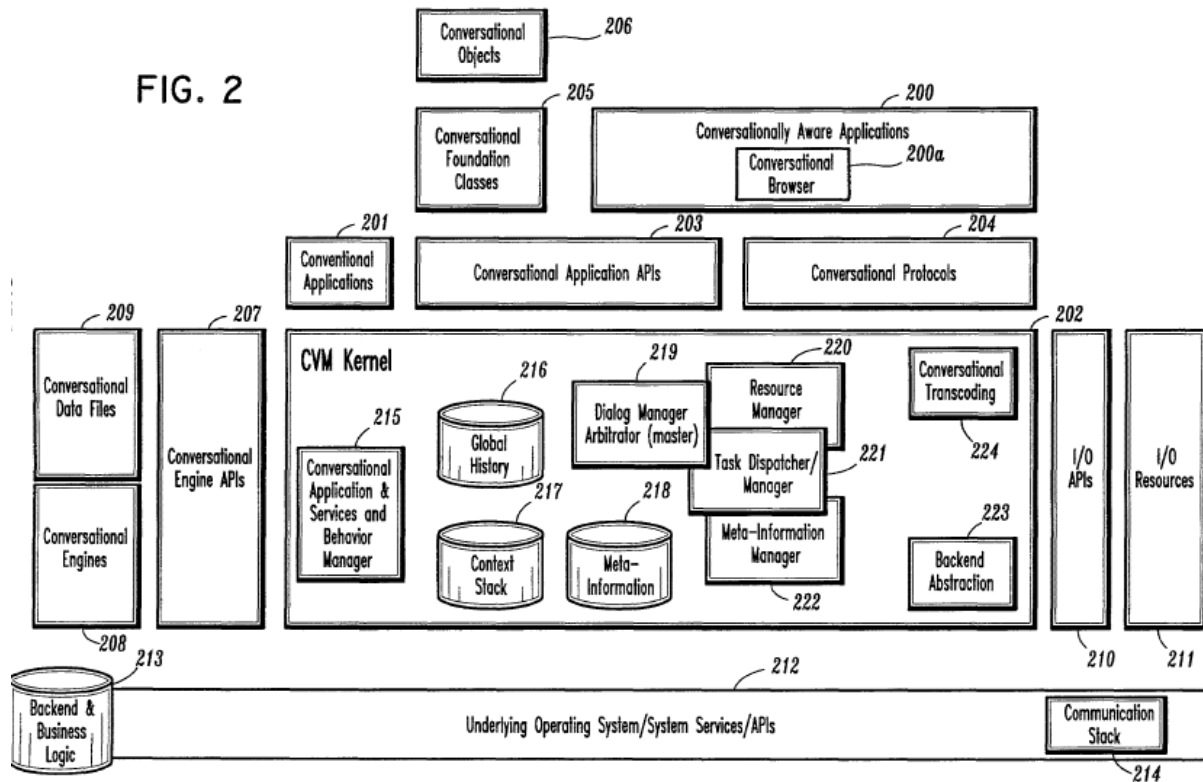
Coffman, 9:27-28.



Per Coffman, the CVM system “comprises a plurality of applications including conversationally aware applications 11 (i.e., applications that ‘speak’ conversational protocols) and conventional applications 12.” Coffman, 12:26-28. The difference between these types of applications, per Coffman, is that “conversationally aware applications 11 are applications that are specifically programmed for operating with a CVM core layer (or kernel) 14 via conversational application APIs 13.” *Id.*, 12:28-30. Conventional applications interact with the CVM, but do not need to do so through the conversational application APIs 13. Thus, for both conventional applications and conversationally-aware applications, the “CVM kernel 14 controls the dialog across applications and devices on the basis of their registered conversational capabilities and requirements[,] and provides a unified conversational user interface ... to provide conversational system behaviors.” *Id.*, 12:30-13:1. In particular, Coffman’s conversationally-aware applications 11 are directly connected to the User Interface 10 to receive a user’s utterance, and interface with the CVM 14 via the conversational application API.

Indeed, the conversational application API serves as an intermediary between the conversationally-aware applications on the frontend, and the backend system including CVM 14, conversational sub-systems 18, and backend logic. The CVM and its components are illustrated and described in more detail with respect

to Coffman's FIG. 2, reproduced below. The CVM Kernel 202 in FIG. 2 corresponds to the CVM Kernel 14 illustrated in FIG. 1.



Relevant to the current Petition, the CVM Kernel 202 includes a Context Stack 217, a Dialog Manager Arbitrator (master) 219 (which is frequently referred to in Coffman as the Dialog Manager 219), and a Backend Abstraction 223. There is also a “conversational application and behavior/service manager layer 215,” which “comprises functions for managing the conventional and conversationally aware applications 200 and 201.” *Id.*, 19:30-31. Coffman describes these management functions as including “for example, keeping track of which applications are registered (both local and network-distributed), what are the dialog

interfaces (if any) of the applications, and what is the state of each application.”

Id., 19:31-20:2.

Describing the context stack 405, Coffman discloses that “[t]he core of the CVM 401 is the context stack 405 which operates and is managed under the control of the dialog controller 404.” *Id.*, 29:19-20. Coffman defines the term “context” to refer “to the state of each discourse (whether active or nonnative), which keeps track of the past history of the discourse, its current state, and the specific characteristics and full query arguments of the corresponding task (e.g[.], vocabulary file, language model, parsing, tags, voiceprint, TTS rules, grammar, NLU etc. of each active task/process) along with any data needed for input understanding.” *Id.*, 29:27-32.

The application-specific “context stack 217 is managed by the dialog manager 219.” *Id.*, 21:23; *see also id.*, 27:1-2 (disclosing that each application will have its “own ... context stack ...”). “More specifically, each new task, process, or thread creates a new stack entry and is associated with a discourse.” *Id.*, 30:4-5. “Each context associated with a given discourse comprises the latest requests made to the corresponding process/task/thread as well as the latest output.” *Id.*, 30:6-8.

The dialog manager 219 (or dialog controller 404) “manages the context stack 405 by creating a new stack entry in the context stack 405 for each new task/process/thread that is spawned either local or remotely from a networked

device.” *Id.*, 30:13-15. “The context stack 405 is associated with the machine state stack so that for any new input from a user, the dialog controller 404 may traverse the context stack 405 until the input context can be appropriately established. This essentially amounts to finding and selecting the active discourse between the user and machine among the last and past discourses.” *Id.*, 30:22-25; *see also id.*, 41:9-13.

“Each application may be associated with multiple discourses (e.g. the application management discourse and the application content navigation discourses).” *Id.*, 29:4-8. An application’s context stack 217 “comprises all the information associated with an application.” *Id.*, 21:23-24. “Such information includes all the variable, states, input, output and queries ***to the backend*** that are performed in the context of the dialog and any extraneous event that occurs during the dialog.” *Id.*, 21:24-26 (emphasis added). The CVM Kernel 202 also includes a “global history 216,” which “includes information that is stored beyond the context of each application,” such as “the information that is associated with all the applications and actions [taken] during a conversational session (i.e., the history of the dialog between user and machine for a current session (or from when the machine was activated).” *Id.*, 21:28-32.

Importantly, Coffman’s mention in this passage of “queries to the backend” is explained further by the inclusion of Coffman’s “backend abstraction layer 223”

in the CVM Kernel 202. The “backend abstraction layer 223 ... allows access to backend business logic 213 via the dialog manager 219 (rather than bypassing the dialog manager 219).” Coffman, 22:1-3. The “backend abstraction layer 223 can translate input and output to and from the dialog manager 219 to database queries. This layer 223 will convert standardized attribute value n-uples into database queries and translate the result of such queries into tables or sets of attribute value n-uples back to the dialog manager 219.” *Id.*, 22:4-7. As Mr. Occhiogrosso notes, Coffman’s backend can process queries formulated at the CVM, and provide the results back to the CVM for communicating them to a conversationally-aware application, which presents the results of the processed query to the user. Occhiogrosso, ¶97.

This understanding of Coffman is also supported by FIG. 4. FIG. 4 of Coffman, reproduced below, provides a block diagram of Coffman’s system 400 and CVM 401 (Coffman, FIG. 4 and at 28:10-12):

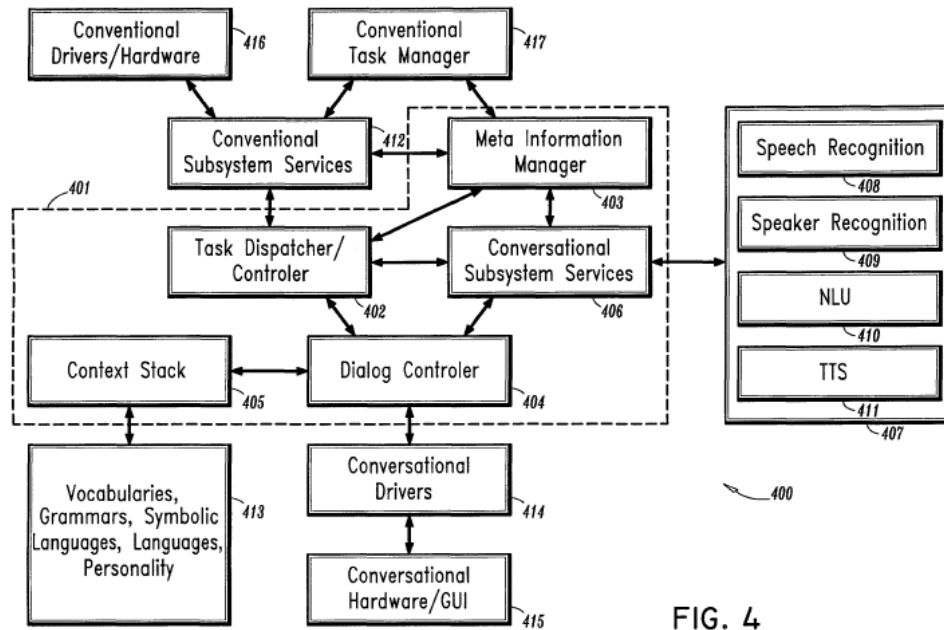


FIG. 4

The conversational system 400 comprises a combination of conventional subsystems and conversational subsystems which are executed and managed by a CVM 401. The CVM 401 comprises a task dispatcher/controller 402, a meta information manager 403, a dialog controller/manager 404, a context stack 405, and a conversational subsystem services manager 406. The “CVM controller” of Coffman is understood to refer collectively to the task dispatcher/controller 402 and the dialog controller 404. In CVM 401, “instructions/commands that are initiated by the user are explicitly *sent to the background* of the system.” *Id.*, 43:1-2 (emphasis added).

This disclosure is critical to a proper understanding of Coffman’s CVM, and how it relates to the conversationally-aware applications disclosed in Coffman. In particular, a user who interacts with a conversationally-aware application on the

front end of Coffman's system will communicate with the conversationally-aware application via speech, and the conversationally-aware application will access the CVM layer (or kernel) via conversational application APIs so that the CVM can perform the backend operations. Occhiogrosso, ¶98. And, as noted, Coffman's CVM and backend operations are hidden from the conversationally-aware applications. *See* Coffman, 5:5-6, 29:10-11. As disclosed by Coffman: "conversational sub-systems 18 are responsible for converting voice requests into queries and converting outputs and results into spoken messages using the appropriate data files 17 (e.g., contexts, finite state grammars, vocabularies, language models, symbolic query maps etc.)." Coffman, 13:8-10.

ii. Coffman's Context Stacks are Unique to Each Application, and Searching an Application's Context Stack Does Not Result in Identifying Another Application to Process a User's Command or Request

Coffman discloses that a CVM may interact with multiple applications. Coffman discloses that the CVM provides applications and devices with "conversational services and behaviors as well as conversational protocols for interaction with *multiple applications* and devices also equipped with a CVM layer, or at least, conversationally aware." Coffman, 11:2-4 (emphasis added). However, Coffman discloses that where there are different applications, "different devices, CVMs or different applications will have their own dialog manager, context stack 217, 217a and global history 218, 218a." Coffman, 27:1-2.

A context stack for an application is further described with respect to FIG. 4's context stack 405, "which operates and is managed under control of the dialog controller 404." *Id.*, 29:18-19. The context stack 405 "accumulates the context (i.e., full query arguments list of attribute value n-uples, and state/mode) of each active process with an activated discourse (i.e., conversational interaction associated with a given task/process/thread) along with any data files 413 (or at least identifiers of such conversational arguments) for the different engines that may be needed for input understanding (e.g., files or arguments that the engines use for performing their respective tasks such as active FSG [Finite State Grammar], topic, vocabulary, HMM (hidden markov models), voiceprints, language models or possible queries for a speech input)." *Id.*, 29:21-27. As such, a search of an application's own context stack 217a may identify an active context and active process for that application. But this search does not support an interpretation that Coffman's CVM would then select and initiate a new process to be performed by a different conversationally-aware application based on the active context. Occhiogrosso, ¶¶98, 108.

iii. Coffman Also Does Not Synchronize Context Stacks or Context Sets

Although Coffman provides a system with context stacks and allows for additions and updates to those context stacks, Petitioner's theories do not identify synchronization of a context stack or context set in a first device with a context

stack or context set in a second device in Coffman. In this regard Coffman is unlike the '652 Patent, which seeks to determine “which version of a data element ... or an agent are the newest or most up-to-date and propagates that element” ('652 pat., 17:26-27), and which can entail “complete replacement,” rather than a series of incremental updates (*id.*, 17:29-32), and, in particular, synchronizes a context stack across different devices (*id.*, 4:29-32 & 19:41-44).

More particularly, Petitioner argues that Coffman synchronizes context stacks, and in support of this argument, cites to portions of Coffman dealing with a device instructing another device to add context entries (*see, e.g.*, Pet. 30-31 & Ex. 1002, ¶¶70 (each citing Coffman, 25:10-14)) and to update context. But Petitioner fails to explain how those activities in Coffman would result in synchronization of context stacks or context sets in the devices.

Coffman's failure to disclose synchronization of context stacks or context sets stands in contrast to its other detailed disclosures regarding what information can be exchanged between the different dialog managers and why that information is exchanged. When describing the protocols for the operation of the different dialog managers in Coffman's system, Coffman lists at least fifteen (15) items of information that *can be—not must be*—exchanged between the different dialog managers in Coffman's system. Coffman, 27:7-22. None of this information is required to be exchanged, and Coffman also fails to disclose any “synchronization”

protocol to synchronize any of these fifteen items among different context stacks in different devices.

Not only does Coffman provide for the non-mandatory exchange of information, Coffman even allows the individual dialog managers the choice to act *or not act* on received information when managing their own context stack, since “different devices, CVMs or different applications ***will have their own ... context stack 217, 217a.***” Coffman, 27:1-2 (emphasis added). For example, Coffman states that the dialog managers provide “notification of I/O events such as ... addition to contexts” (Coffman, 27:12-13 (item 5)), which allows the recipient to act on the notification *without requiring any synchronization* of the entire context stack. Similarly, Coffman allows for “transfer of ... ***proposed*** context changes” (Coffman, 27:16-18 (item 11) (emphasis added)) and data associated with the proposed context changes (Coffman, 27:19-20 (item 15) (“data files, context and state updates due to action”). Coffman’s use of the term “proposed” rather than “required” illustrates that Coffman does not even require any synchronization of individual context items, much less synchronization of the context stack. Items 12 and 14 provide context update and context change notifications but, again, do not require any synchronization of the context stack. *See* Coffman, 27:16-18 (item 12: “decision notification, context update, action update, state update, etc.” & item 14: “notification of context changes”).

B. Petitioner's Ground 1 Fails to Satisfy Synchronizing a First Context Stack (or Context Set) with a Second Context Stack (or Context Set)

Coffman fails to disclose every claim requirement of the challenged '652 patent claims. In particular, claims 1 and 14 of the '652 patent recite a “first context stack associated with a first device” and requires *synchronizing “the first context stack with a second context stack* associated with a second device.”

Similarly, claim 20 recites a “first context set associated with a first device” and “*synchronizing the first context set with a second context set* associated with a second device.”

Petitioner relies on Coffman to assert that Coffman “makes clear” these limitations are satisfied, arguing (Pet. 30-31) (emphasis in original):

Coffman further makes clear that a master device's context stack ... is ‘**synchronized**’ with a slave device's context stack ..., explaining:

With a master/slave network configuration, the master dialog manager can compile all the lists and match the functions and needs with conversational capabilities. In addition, context information may be transmitted by indicating passing or pointing to the context stack/history of the device or application that the controller can access and add to its context stack.

(Coffman, 25:10-14; Schmandt, ¶70.) Coffman further explains that, using dialog manager protocols, devices can exchange

various kinds of information, including notification of “addition to contexts,” “context update” and “notification of context changes.” (Coffman, 27:7-20.)

But as introduced above in Section IV.A.iii, none of these passages address synchronizing a “context stack” or “context set” in one device with another “context stack” (or “context set”) in another device. Occhiogrosso, ¶69. They certainly do not “make clear” that any synchronization is occurring and do not even mention “synchronizing.” *Id.*, ¶¶69-72.

Instead, the actions cited in Coffman indicate that the master dialog manager may instruct a slave dialog manager *to add or update* an entry to its context stack, but Coffman does not disclose or suggest that the stacks themselves would be synchronized as required by the claims of the '652 patent. Occhiogrosso, ¶¶69-72. Petitioner has not shown, or even attempted to explain, that the activities of “addition” to a context stack or updating existing information for a context entry in context stack would necessarily or inherently produce two context stacks that are synchronized. *Id.*

During deposition, Petitioner's expert Mr. Schmandt was asked about two “stacks [with] the same content but in different orders within their stacks,” and whether “a POSITA [would have viewed those stacks as identical?” Mr. Schmandt's response was unequivocal, “No, because order is a – order is an attribute of stack.” Ex. 2011, 13:10-17. Mr. Schmandt's testimony confirms that

for two stacks to be synchronized, it is not enough that they contain the same entries; they also must contain those entries in an identical order because order is an attribute of a stack. *Id.*, *see also id.*, 14:13-16 (Q: “And I think we both agree that order of data within a stack is an attribute of a stack, correct? A: Yes, correct.”); 14:25-15:6 (“So if one stack has different data or more data than the other stack, then they are not identical ...”).

Patent Owner's expert also agrees with this opinion (Occhiogrosso, ¶73), and explains that a POSITA would not interpret two stacks containing different context entries, or even equivalent context entries but in different orders, as being synchronized. Occhiogrosso, ¶¶70, 73-74. As a result, Coffman's disclosure that a master dialog manager may instruct a slave dialog manager *to add* or *update* context in a context stack is not sufficient to conclude that these stacks are “synchronized.” As Patent Owner's expert explains, a POSITA would not interpret the activities of “addition” to a context stack or updating existing information for a context entry in context stack to produce two context stacks that are synchronized. *Id.* Patent Owner's expert continues, noting that a POSITA would understand synchronization to require more than the operations discloses in Coffman. *Id.* For example, since a stack is an ordered list, simply “adding” an entry based on an instruction from another device would not have the effect of

synchronizing two context stacks without knowing that the entry is in the same relative location in both stacks, along with other conditions of synchronization. *Id.*

Indeed, Petitioner’s theory cannot overcome the distinction the Examiner emphasized between the cited art (which included Coffman) and the allowed claims in the ’652 patent—that a showing of even an individual context synchronization does not show that “context stacks/sets of distinct devices are synchronized.” Ex. 1012, 447 (emphasis in original).

Moreover, not only do Coffman’s passages at 25:10-14 and 27:7-20 fail to “make clear” that Coffman possesses synchronous context stacks in different devices, but Coffman itself contradicts Petitioner’s theory when stating that “different devices, CVMs or different applications ***will have their own ... context stack 217, 217a.***” Coffman, 27:1-2. As noted above, the protocols for information exchange between different dialog managers propose, rather than require, context changes (Coffman, 27:16-18 (item 11)), making Petitioner’s unsupported leap—from Coffman’s instruction to add or update a context entry, to the synchronization of a context stack or context set—untenable. Occhiogrosso, ¶75.

Also, Petitioner does not present any argument or expert testimony concerning the desirability of synchronized context stacks in Coffman. In providing testimony regarding the “synchronize” requirement of the challenged claims, Petitioner’s expert merely parrots the language of the Petition (*compare*

Pet. 30-31 *with* Ex. 1012, ¶70) without providing any basis for the opinion presented. Petitioner's expert and does not state any benefit, use, or improvement that would be achieved to Coffman if synchronized context stacks were present in Coffman (which they are not), or any additional reasoning why Coffman's disclosure would lead a POSITA to understand that Coffman discloses or suggests the required synchronizing of context stacks or context sets. In the face of such unsupported and conclusory testimony, Petitioner's argument and its expert's testimony should not be relied upon. *Consolidated Trial Practice Guide* at 36 (Nov. 2019) ("expert testimony cannot take the place of disclosure from patents or printed publications"); 37 C.F.R. § 42.65(a). As Patent Owner's expert explains, nothing in Coffman's protocols addresses the order of a target context stack based on a source context stack, such that both the source context stack is synchronized with the target context stack. Occhiogrosso, ¶¶69-73. Coffman is similarly deficient with regard to synchronization of context sets. *Id.*

Thus, the Petition fails with regard to Ground 1 and independent claims 1, 14, and 20, along with corresponding dependent claims 2-13, 15-19, and 21-27. Grounds 2 and 3, which are dependent on the success of Ground 1's challenges to the independent claims, also necessarily fail, particularly as Petitioner does not rely on Kennewick or Lee to remedy the deficiencies of Coffman with regard to the claimed synchronization.

C. Petitioner's Ground 1 Also Fails to Satisfy the Identification of Domain Agents Configured to Process a Command or Request of Dependent Claims 8, 18, and 24

Coffman also fails to disclose every claimed requirement of at least dependent claims 8, 18, 24, and their respective dependent claims. Claim 8 of the '652 patent depends from claim 1 via claim 3 and recites, in relevant part, one or more physical processors caused to “identify one or more domain agents associated with the one or more context entries, wherein the one or more domain agents are configured to process the command or request; and generate a response to the command or request using the one or more domain agents.” As Petitioner contends, claims 18 and 24 include substantially similar features as claim 8, although they depend from different base claims. Pet., 59, 60.

Petitioner alleges that Coffman's “executable software of a conversationally-aware application (and associated data, such as vocabulary, grammar, language models and NLU symbolic mapping)” discloses the claimed “domain agent” of claim 8. Pet., 54. Petitioner also alleges that Coffman's conversationally-aware application is “associated with the one or more context entries” of claim 8. Pet., 55. These assertions are incorrect, and reflect Petitioner's fundamental misunderstanding of Coffman.

Specifically, Petitioner's characterization of Coffman's conversationally-aware application as a “domain agent” is unsupported by Coffman, and overlooks

that Coffman's conversationally-aware applications are gateways to use the system, providing an interface (UI) for a user to provide the speech-based content for a given conversationally-aware application to interact with users. A conversationally-aware application in Coffman accesses the tools provided by Coffman's conversational virtual machine ("CVM") via Coffman's application program interfaces (APIs), and awaits a response from the CVM to continue a communication with a user. That is all that Coffman's conversationally-aware applications do, and it not at all comparable to what the claimed "domain agent" is required to do when claim 8 is viewed in its entirety.

Contrary to Petitioner's assertion, Coffman's CVM does not identify any conversationally-aware application, based on an identified context entry, configured to process a command or request, and does not generate a response to the command or request using the one or more domain agents. Indeed, in Coffman it is a user who "selects" a conversationally-aware application to be used. Occhiogrosso, ¶¶85, 98, 111. In response to a particular conversationally-aware application making an API call containing a user's utterance, Coffman's CVM activates a processing thread that accesses a context stack specific to that conversationally-aware application, identifies the active discourse (or conversation) from that context stack, and carries out a process associated with the active discourse's context. There is no identifying a conversationally-aware

application based on an identified context entry, and Coffman's conversationally-aware application is not configured to process a command or request. *Id.*

Instead, Coffman searches for an active context in a stack of contexts by comparing "new queries or arguments" to the stack "until an acceptable match is obtained and optionally confirmation is obtained from the user." Coffman, 42:1-

7). But Coffman's determination of an active context does not lead to an identification of any conversationally-aware application as a "domain agent."

Rather, the search for an active context follows the user's selection of, and interaction with, a conversationally-aware application and the identification of that application's corresponding context stack.

Relevant to this point, Coffman's disclosure at 42:1-7 is reproduced below:

Completion/search for the active context is performed from context to context down the stack. That is, new queries or arguments are compared by the dialog engine by going down the stack until an acceptable match is obtained and optionally confirmation is obtained from the user. As soon a context is found that fits the utterance at the level of the NLU symbolic language, the *context becomes active and the corresponding process becomes active*. Until the active command is completed, or until a new command is provided, the selected context is marked active, and pushed to the top of the context stack 405.

Coffman, 42:1-7 (emphasis added). Regarding the “corresponding process” disclosed in this passage, and contrary to Petitioner’s assumption, Coffman does not disclose that the “process” in the passage above occurs at a “conversationally-aware application,” and does not disclose that activating a process is the same as selecting a conversationally-aware application. Occhiogrosso, ¶¶95, 104. Indeed, Coffman associates a “process” with tasks and threads performed *at the CVM*. See Coffman, 42:16-17 (“This stack structure allows to maintain non-ambiguous conversational interactions with multiple tasks, threads or processes.”); 21:1-3 (“The task dispatcher/manager 221 dispatches and coordinates different tasks and processes that are spawned (by the user and machine) on local and networked conventional and conversational resources”); 30:1-2 (“context stack (as well as other CVM components)”); 30:4-5 (“[E]ach new task, process, or thread creates a new stack entry and is associated with a discourse.”); FIG. 2 (illustrating CVM components including task dispatcher/manager 221). This disclosure confirms that Coffman processes user requests and commands at the CVM, and that Coffman does not select a conversationally-aware application to process a user request and command, or generate a response to a user request or command. Occhiogrosso, ¶¶95, 98, 104.

Nothing about this disclosure suggests that the CVM identifies a conversationally-aware application for processing a user’s request or command.

To the contrary, Coffman expressly discloses that the task dispatcher/controller 402 of the CVM 401 “dispatches and coordinates different tasks and processes that are spawned” Coffman, 30:26-28. As Mr. Occhiogrosso notes, nothing in Coffman’s discussion of the task dispatcher/controller 402 suggests or discloses identifying a conversationally-aware application based on a match in the context stack, or even that the task dispatcher/controller 402 is capable of identifying a conversationally-aware application. Occhiogrosso, ¶105. Instead, Coffman selects a context stack in response to the user’s interaction with a conversationally-aware application.

Moreover, Coffman discloses that “all conversational subsystem calls are hidden to the application” Coffman, 29:10-11; *see also id.*, 5:5-6. The CVM 401, not any application, “will locate the requested conversational subsystem 407 (via the conversational subsystem services manager 406), drive its execution and return appropriately the *results*” to the application. *Id.* (emphasis added). It is the CVM 401 that includes the “context stack 405.” *Id.* As Patent Owner’s expert explains, this disclosure confirms that Coffman processes user requests and commands at the CVM, and that Coffman does not select a conversationally-aware application to process a user request and command, or generate a response to a user request or command. Occhiogrosso, ¶106. Thus, Petitioner is incorrect that identification of a context in a context stack yields the identification of a

conversationally-aware application to process a user's request or command.

Instead, the subsystems are invisible to the conversationally-aware application, which receives the final results to be reported to the user. Coffman fails to disclose the claimed "identify one or more domain agents" feature.

Confirming that Coffman's determination of an active context in the context stack does *not* lead to an identification of a conversationally-aware application based on an identified context entry, Coffman discloses that where there are different applications, "different devices, CVMs or different applications will have their own dialog manager, context stack 217, 217a and global history 218, 218a." Coffman, 27:1-2. Similarly, Coffman explains that a "context stack 217 comprises all the information associated with *an application*." Coffman, 21:23-24 (emphasis added). As such, a search of an application's own context stack 217 may identify an active context and active process for that application, but does not support Petitioner's expansive reading that Coffman's CVM will then identify a conversationally-aware application among many applications to process a user's request based on the context stack search. Occhiogrosso, ¶108. Specifically, a context stack for one application does not direct Coffman's CVM to call another conversationally-aware application to generate a response to a user's command or request. *Id.*

Rather, a user interacting with a conversationally-aware application in Coffman results in, at most, an identification of an active context for that user's current discourse within that application. *See* Coffman, 30:4-5 (“[E]ach new task, process, or thread creates a new stack entry and is associated with a discourse.”); 29:4-5 (“Each application may be associated with multiple discourses”); 30:23-24 (“[F]or any new input from a user, the dialog controller 404 may traverse the context stack 405 until the input context can be appropriately established. This essentially amounts to finding and *selecting the active discourse between the user and machine* among the last and past discourses.”) (emphasis added). This does not disclose or suggest an identification of an application configured to process a user's request or command by Coffman's CVM, contrary to Petitioner's theory. Occhiogrosso, ¶109. That theory is wrong.

Coffman also fails to disclose “generate a response to the command or request using the one or more domain agents,” as claimed. Petitioner again alleges that Coffman “discloses that a conversationally-aware application returns output as a result of processing the command or request, which is used to generate a response to provide to the user.” Pet., 56 (citing Schmandt, ¶117). This is also incorrect.

A correct understanding of Coffman would recognize that a user initiates interaction with a conversationally-aware application in Coffman, and that a

conversationally-aware application is neither selected or later invoked based on the determined meaning of an utterance. Occhiogrosso, ¶¶110-11. Moreover, generating a response to the command or request does not occur within Coffman's conversationally-aware application. *Id.* Nowhere in the discussion of Coffman is there a selection of an application, such as a calendar application, based on a determined meaning of a user's utterance. *Id.* Any selection of an application is by a user who chooses to interact with a conversationally-aware application, not by the CVM based on a context entry identification. *Id.* Coffman's identification of a context entry or even activation of a process does not then lead to an application selection, as the Petition tries to manufacture. *Id.*

Petitioner also cites to Coffman's 13:8-13 and 58:9-11 in support of its theory for dependent claim 8. Pet., 57. But these disclosures do not support Petitioner's theory because Coffman's disclosures at 13:8-13 and 58:9-11 describe actions at the CVM 14, not a conversationally-aware application. *See* Coffman, 13:8-13, 58:9-11; Occhiogrosso, ¶112.

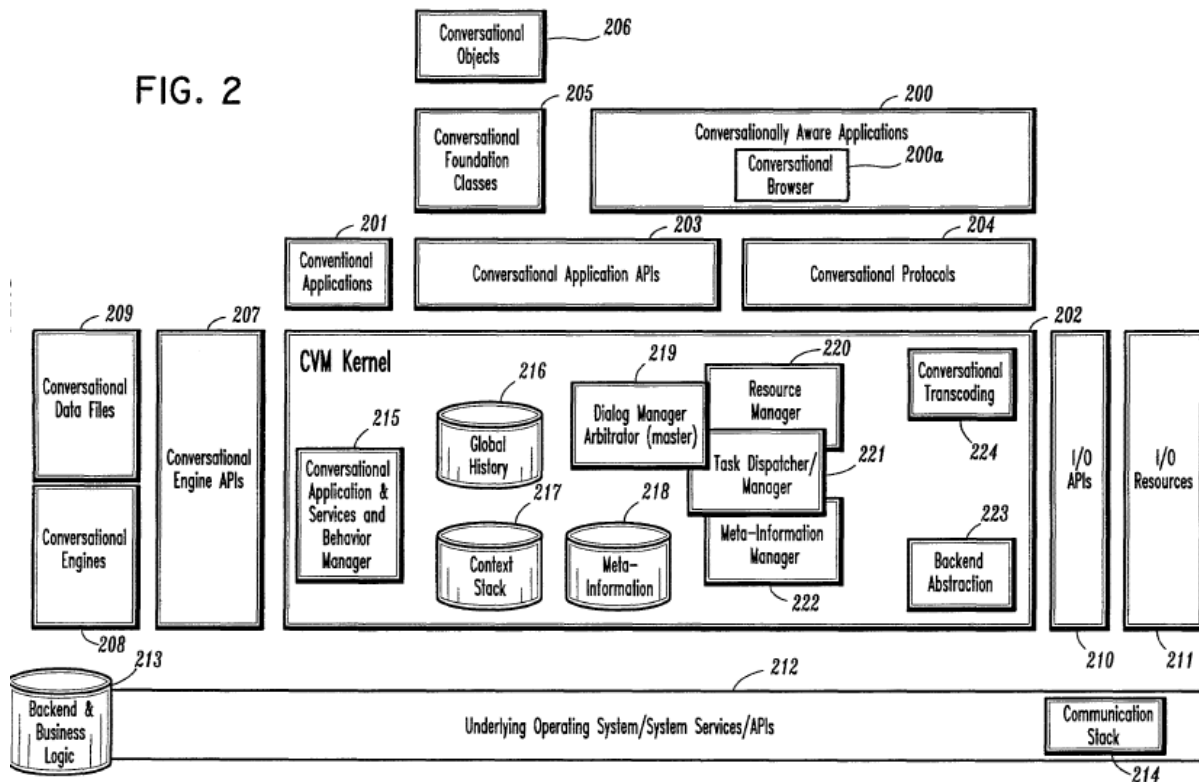
Coffman's disclosure at 30:4-10 explains that "each new task, process, or thread creates a new stack entry and is associated with a discourse," and that each "application may be associated with multiple discourses" Coffman, 30:4-6.

But Coffman also discloses that each application has its own context stack,⁵ and Coffman's description of the context stack contents for various discourses for an individual application does not support Petitioner's theory that a conversationally-aware application is identified to process a user's request. Occhiogrosso, ¶113. The creation of a new stack entry in a specific application's context stack does not support the invocation of another application based on a determined meaning of an utterance or an identified context entry corresponding to a user's words. *Id.*

Further, Coffman's conversationally-aware application is not configured to process a user's request or command, as claimed. As already discussed above, Coffman explains that "all conversational subsystem calls are hidden to the application" Coffman, 29:10-11. The CVM 401, not any application, "will locate the requested conversational subsystem 407 (via the conversational subsystem services manager 406), drive its execution and return appropriately the *results*" to an application. *Id.* (emphasis added).

This process is shown and described with Coffman's FIG. 2. There, Coffman's conversationally-aware application communicates with the CVM Kernel 202, including the dialog manager 219, backend abstraction layer 223, and backend business logic 213.

⁵ Where there are different applications, "different devices, CVMs or different applications will have their own dialog manager, context stack 217, 217a and global history 218, 218a." Coffman, 27:1-2.



Coffman's "backend abstraction layer 223 ... allows access to backend business logic 213 via the dialog manager 219 (rather than bypassing the dialog manager 219)." Coffman, 22:1-3. The "backend abstraction layer 223 can translate input and output to and from the dialog manager 219 to database queries. This layer 223 will convert standardized attribute value n-uples into database queries and translate the result of such queries into tables or sets of attribute value n-uples back to the dialog manager 219." *Id.*, 22:4-7. Coffman's conversationally-aware applications rely on the CVM, including dialog manager 219 and backend business logic 213, to implement its tools and generate a response to a user's utterance. The processing of user questions, queries and commands is performed

by the CVM and the backend subsystems, not by Coffman's conversationally-aware applications. Occhiogrosso, ¶117.

Thus, Petitioner is incorrect that identification of a context entry in a context stack yields the identification of a conversationally-aware application configured to process a user's request or command. The conversationally-aware application receives processed results for presenting to the user from the CVM and backend, but is not invoked to process those results.

The Petition is wrong in trying to establish that Coffman's conversationally-aware applications disclose the claimed "one or more domain agents" of dependent claims 8, 18, and 24. Occhiogrosso, ¶118. For these reasons, Petitioner has failed to establish the unpatentability of dependent claims 8, 18, and 24, and their respective dependent claims 9-11 and 25-27.

V. CONCLUSION

For the reasons presented above, the Petition fails to establish the unpatentability of any challenged claim. Patent Owner asks that the Board issue a final written decision confirming patentability of each challenged claim.

Dated: February 21, 2025

/Wayne M. Helge/

Wayne M. Helge, Reg. No. 56,905

James T. Wilson, Reg. No. 41,439

BUNSOW DE MORY LLP

277 S. Washington St., Suite 210 #1088

IPR2024-00752 Patent Owner's Response

Alexandria, VA 22314

T: (571) 208-0186

Email: whelge@bdiplaw.com

Email : jwilson@bdiplaw.com

Counsel for Patent Owner

CERTIFICATE OF WORD COUNT

The undersigned certifies that the foregoing PATENT OWNER'S PRELIMINARY RESPONSE complies with the type-volume limitation in 37 C.F.R. § 42.24(b)(1). According to the word-processing system's word count, the brief contains 9,286 words, excluding the parts of the brief exempted by 37 C.F.R. § 42.24(a).

By: /Wayne M. Helge/
Wayne M. Helge (Reg. No. 56,905)
Attorney for Patent Owner

CERTIFICATE OF SERVICE

I hereby certify that on February 21, 2025, a true and correct copy of the foregoing document was served via email, by consent, to Petitioner by serving the correspondence email addresses of record as follows:

Heidi L. Keefe
hkeefe@cooley.com

Andrew C. Mace
amace@cooley.com

Mark R. Weinstein
mweinstein@cooley.com

Reuben Chen
rchen@cooley.com

COOLEY LLP
ATTN: Patent Group
1299 Pennsylvania Ave. NW, Suite 700
Washington D.C. 20004
Tel: (650) 842-7800

/Wayne M. Helge/
Wayne M. Helge (Reg. No. 56,905)
Attorney for Patent Owner