

(54) **ARTIFICIAL-INTELLIGENCE-ASSISTED
ERROR PREDICTION IN INTEGRATION
PROCESSES**

(71) Applicant: **Boomi, LP**, Conshohocken, PA (US)

(72) Inventors: **Aditi Paul**, Jersey City, NJ (US);
Ching-Han Tu, San Diego, CA (US);
Bradley Detlefsen, West Chester, PA
(US)

(73) Assignee: **Boomi, LP**, Conshohocken, PA (US)

(21) Appl. No.: **18/438,244**

(22) Filed: **Feb. 9, 2024**

Publication Classification

(51) **Int. Cl.**
G06F 11/00 (2006.01)
G06F 40/40 (2020.01)
G06N 3/0455 (2023.01)

(52) **U.S. Cl.**
CPC **G06F 11/004** (2013.01); **G06F 40/40**
(2020.01); **G06N 3/0455** (2023.01); **G06F**
2201/81 (2013.01)

(57) **ABSTRACT**

Conventional error detection for integration processes in an integration platform are inefficient and require significant expertise. Accordingly, an error prediction model is disclosed. The error prediction model may be operated to produce error predictions, based on the current design (e.g., lineage) of an integration process, during construction of that integration process (e.g., on a virtual canvas). A generative language model may also be used to provide the error predictions in natural language. This enables the efficient troubleshooting and resolution of errors in an integration process, prior to that integration process being deployed and executed, and without requiring significant expertise.

100

The diagram illustrates a system architecture (100) for error prediction in integration processes. A cloud environment (140) contains a process flow (160) and an API (162). The cloud (140) is connected to a Platform (110) which includes an AI Model (114), a Database (116), and a Server Application (112). The Platform (110) is connected to a Network(s) (120) via a router (150). The Network(s) (120) is connected to a Third-Party System (170) and a User System (130).

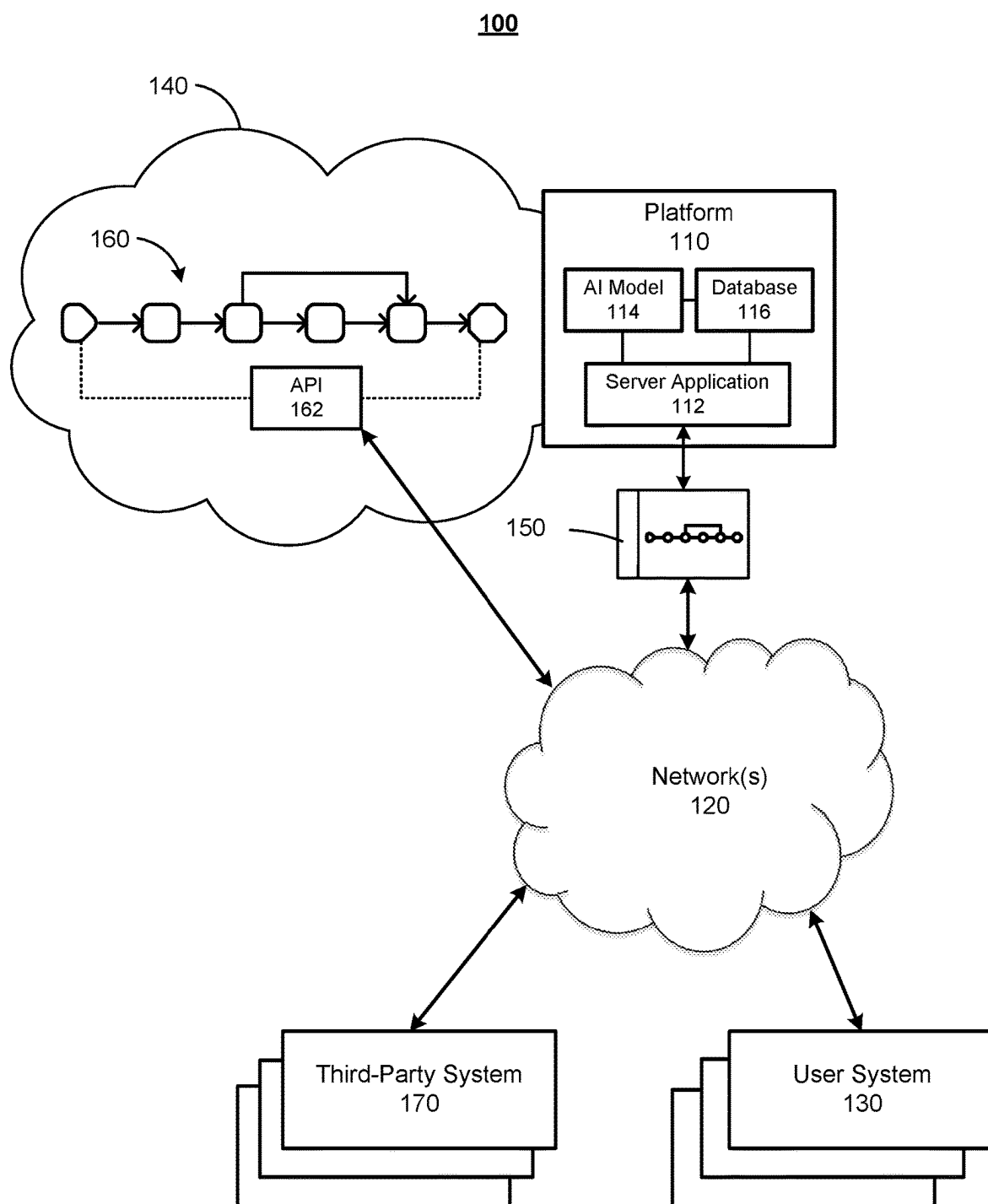
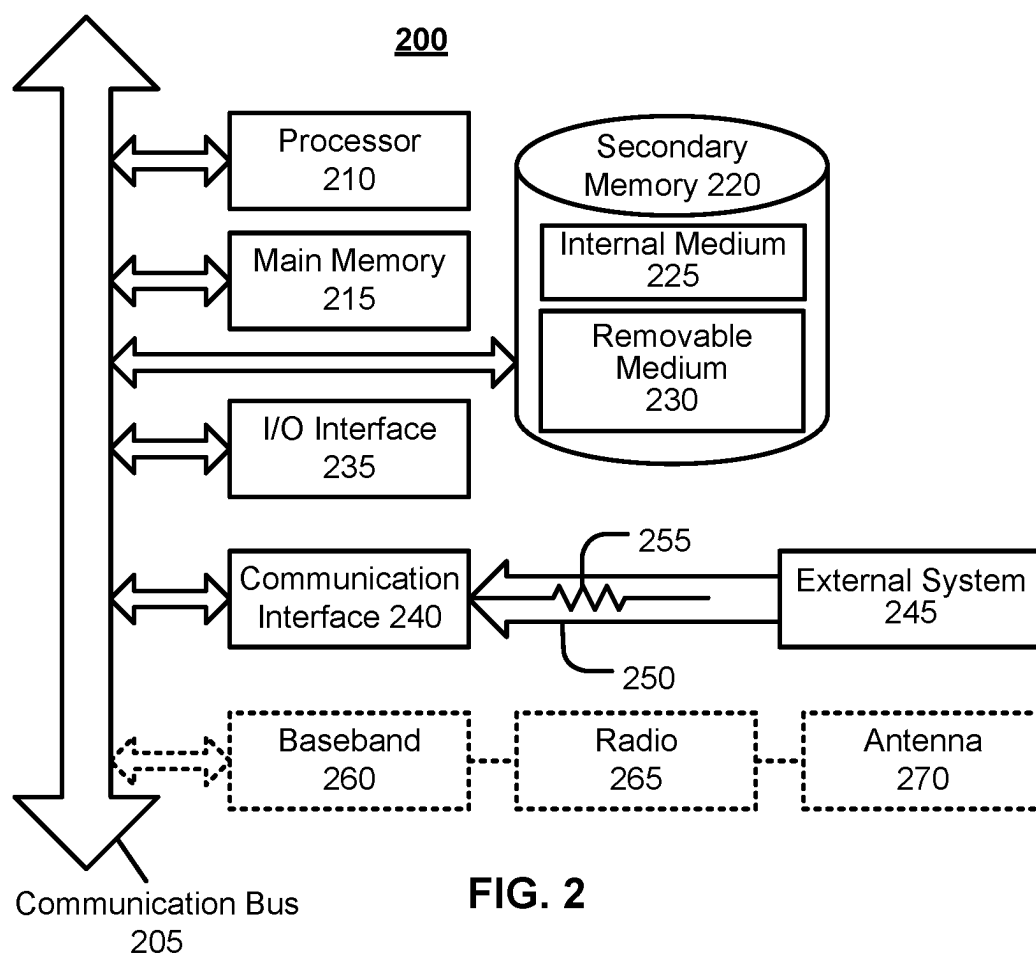


FIG. 1



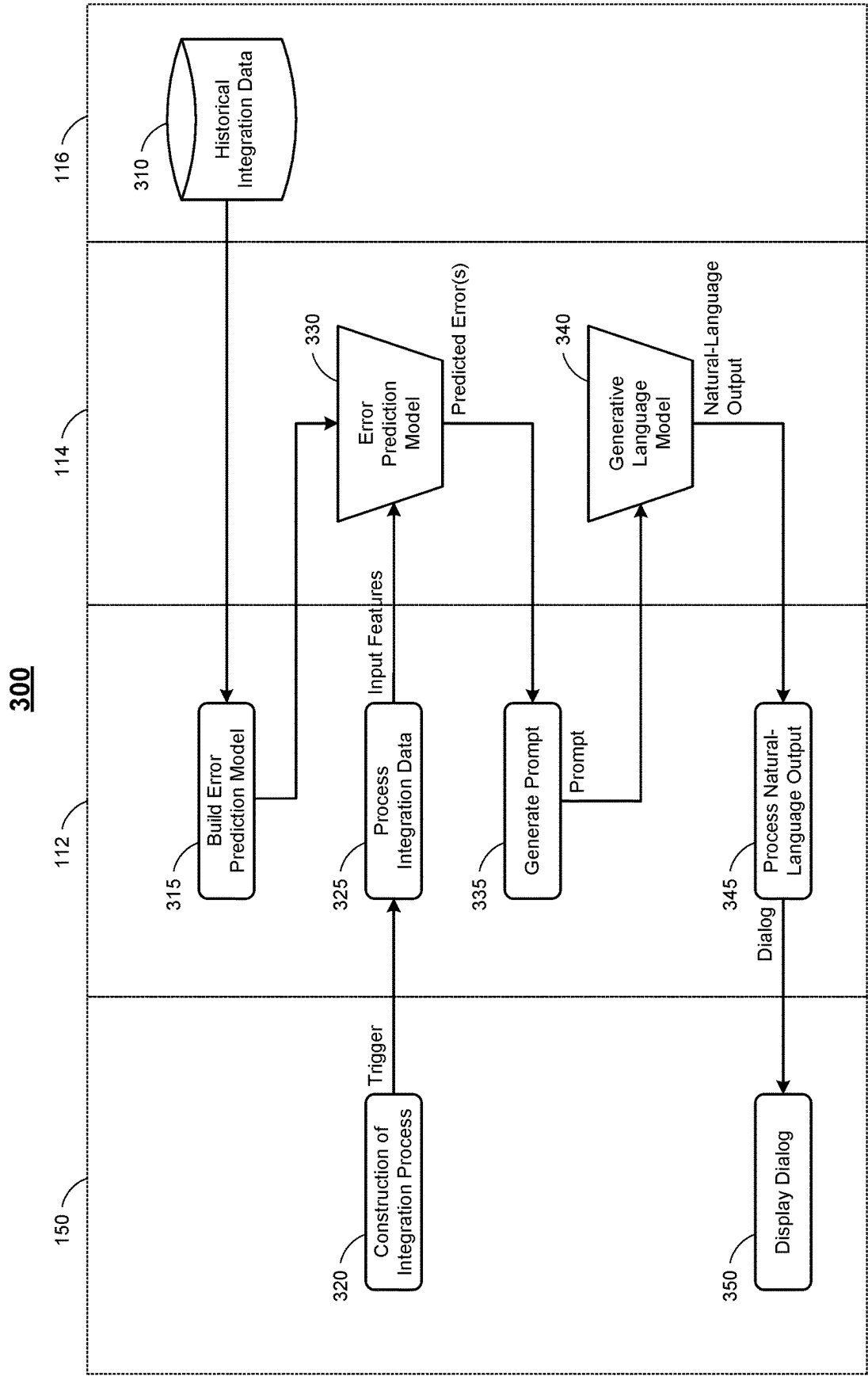


FIG. 3

FIG. 4

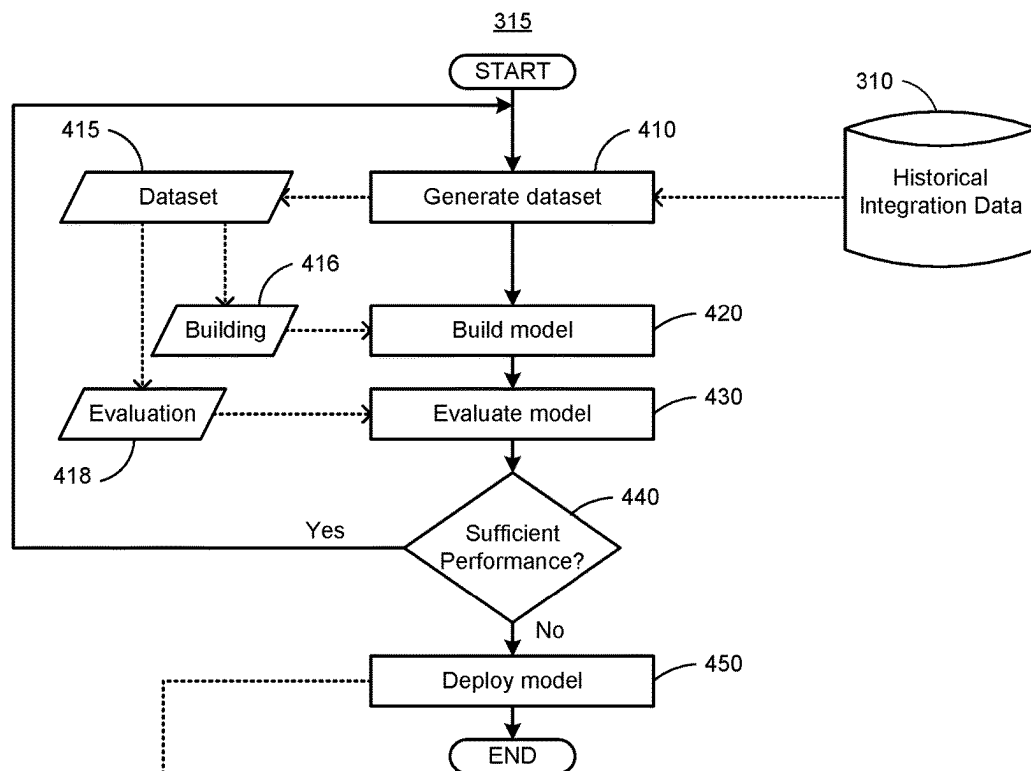
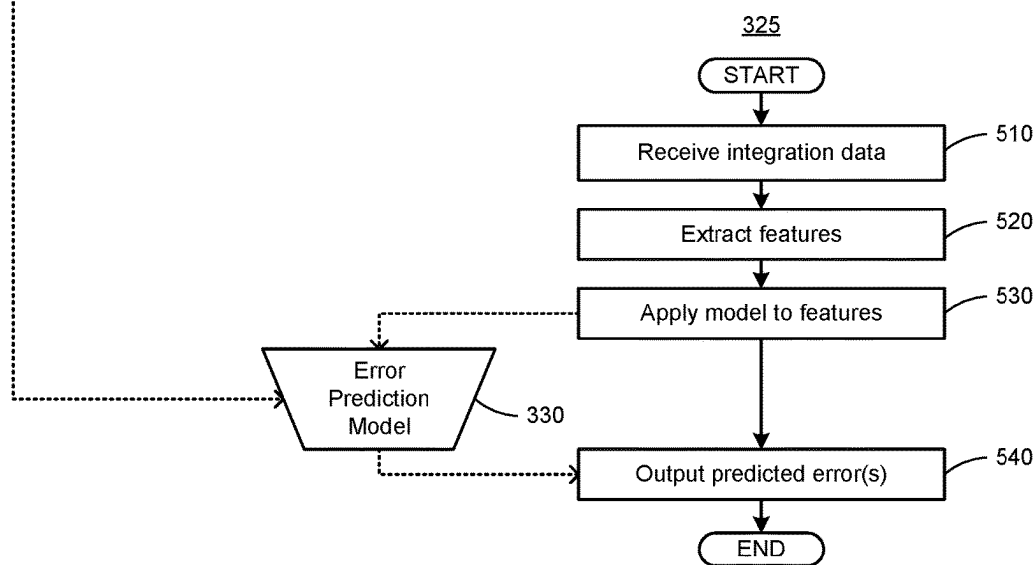
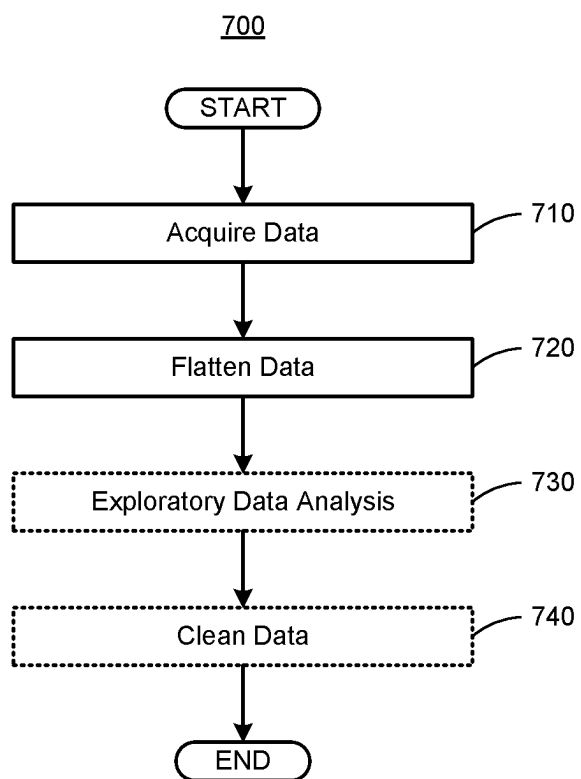
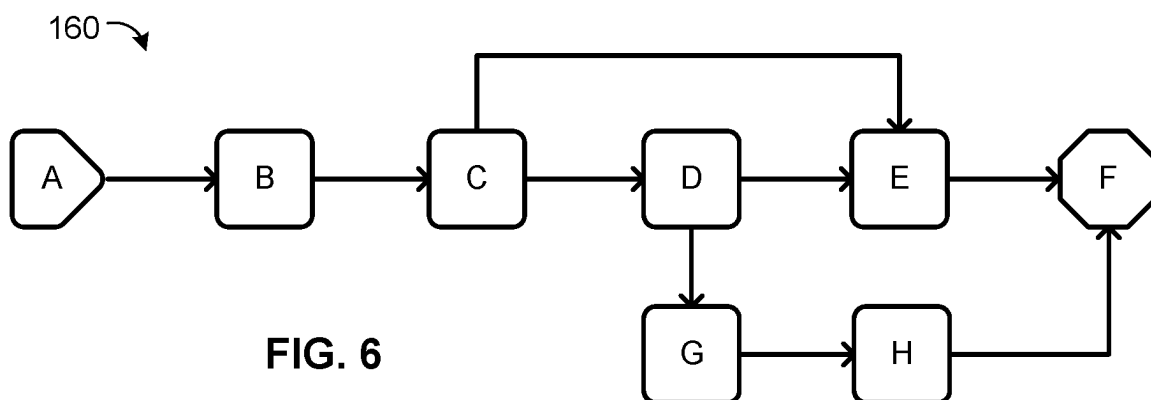


FIG. 5





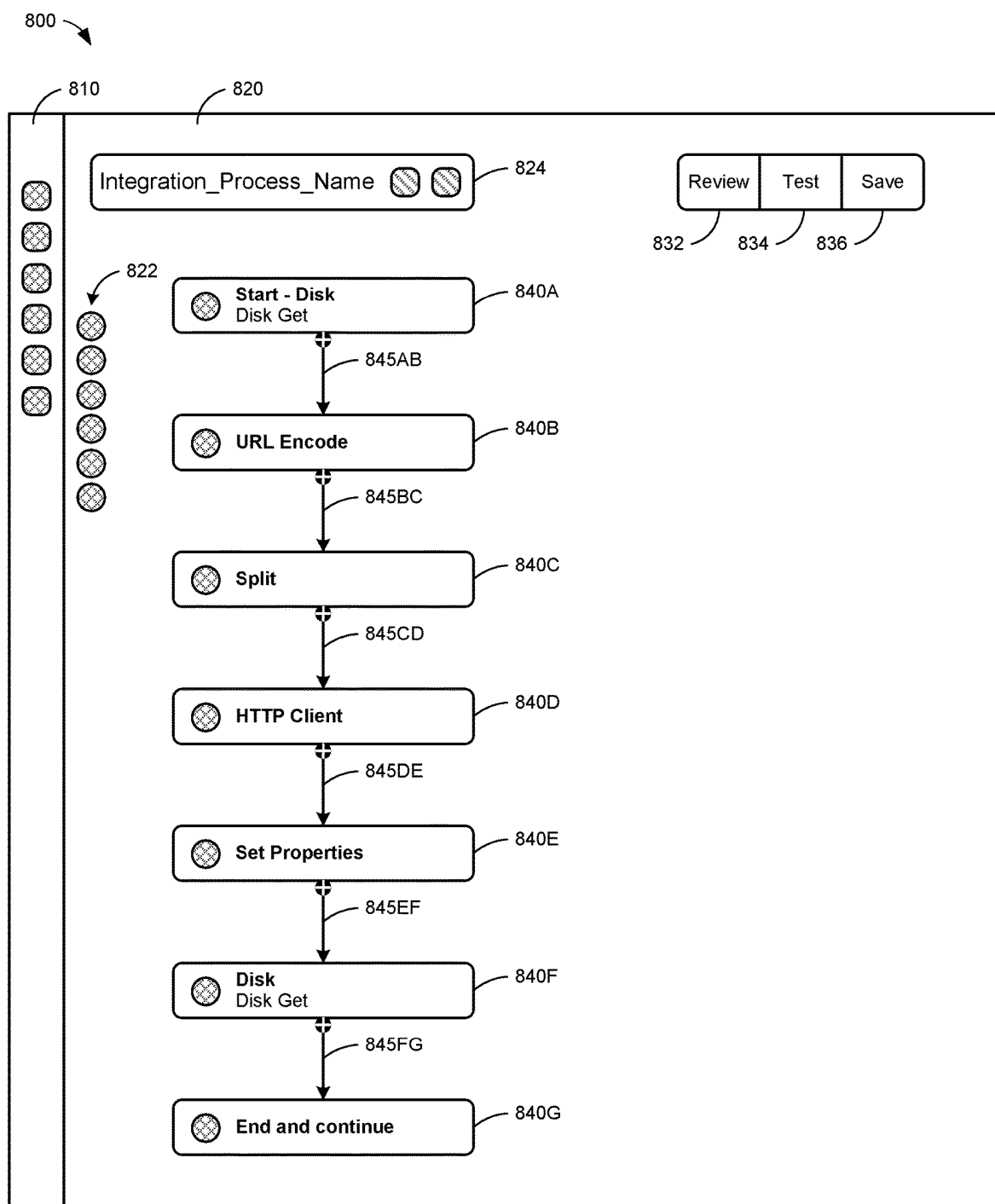


FIG. 8A

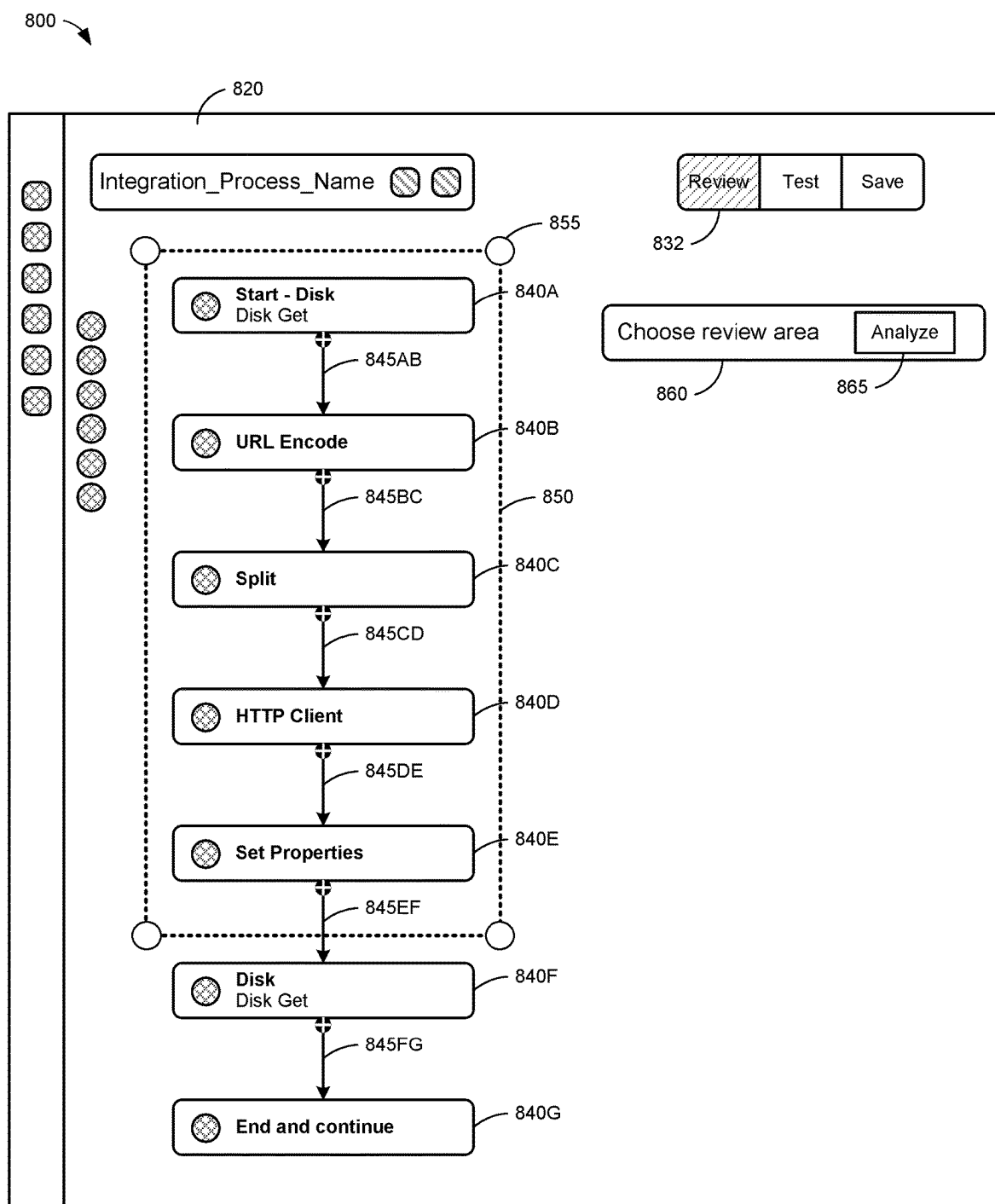


FIG. 8B

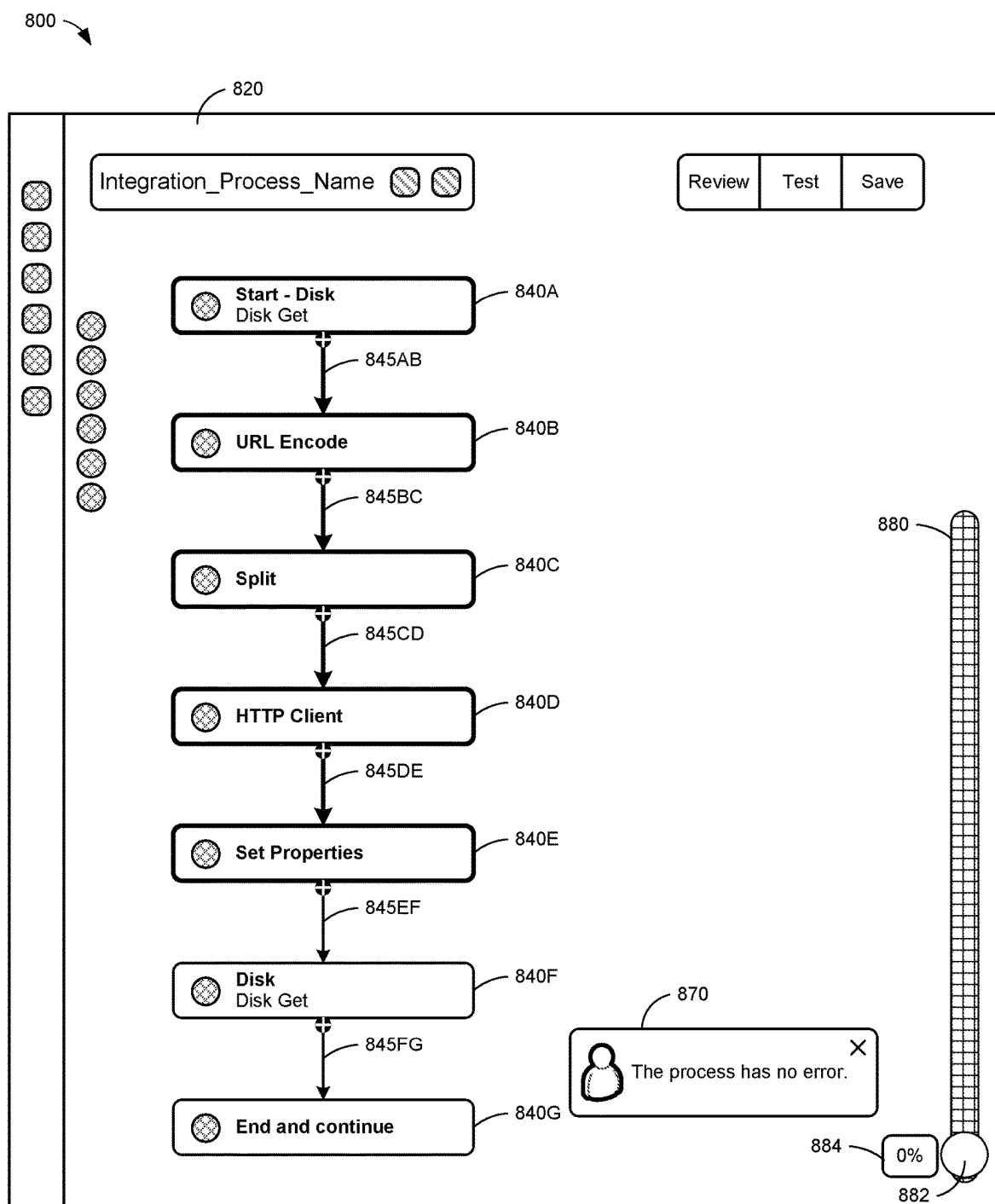


FIG. 8C

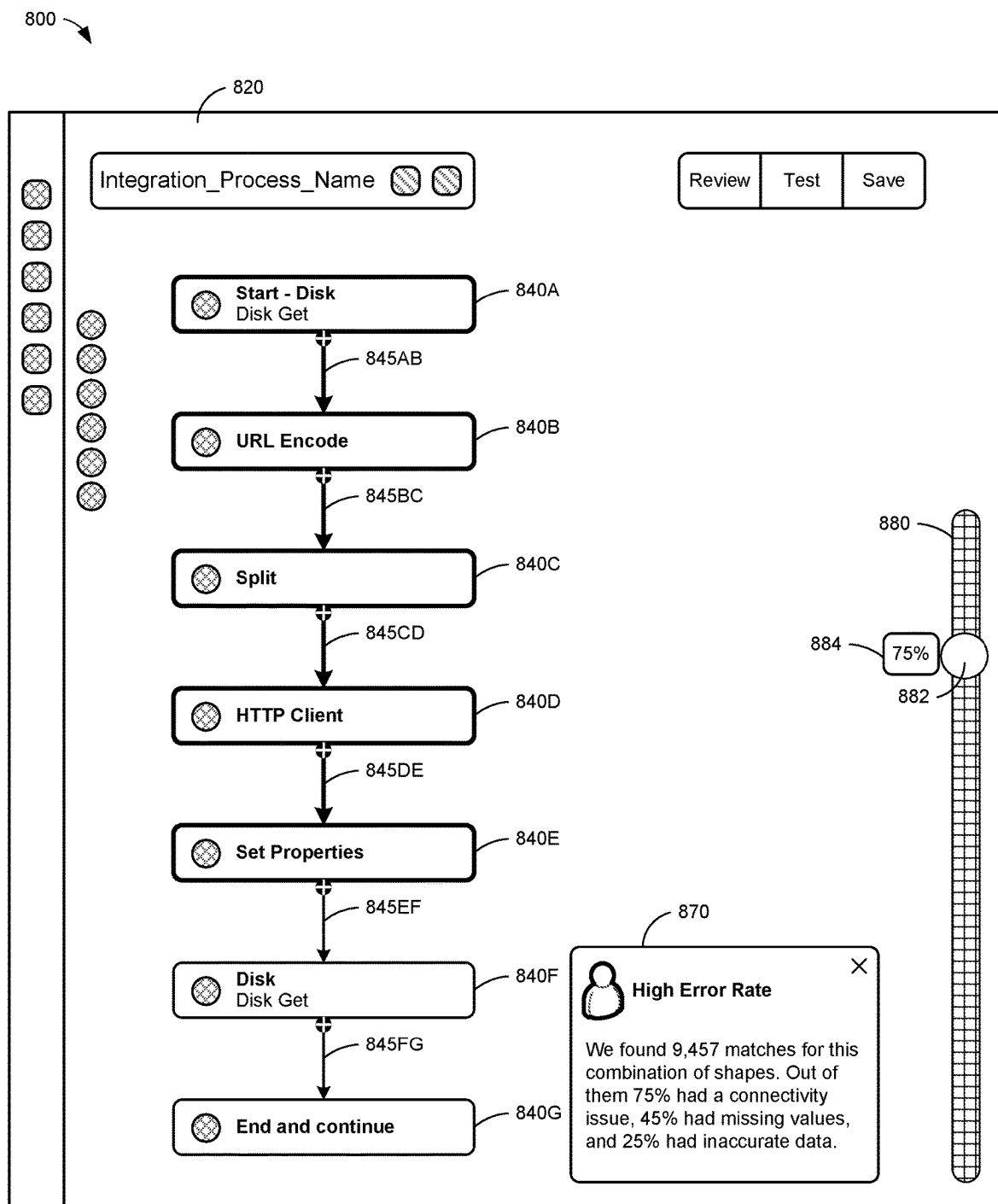


FIG. 8D

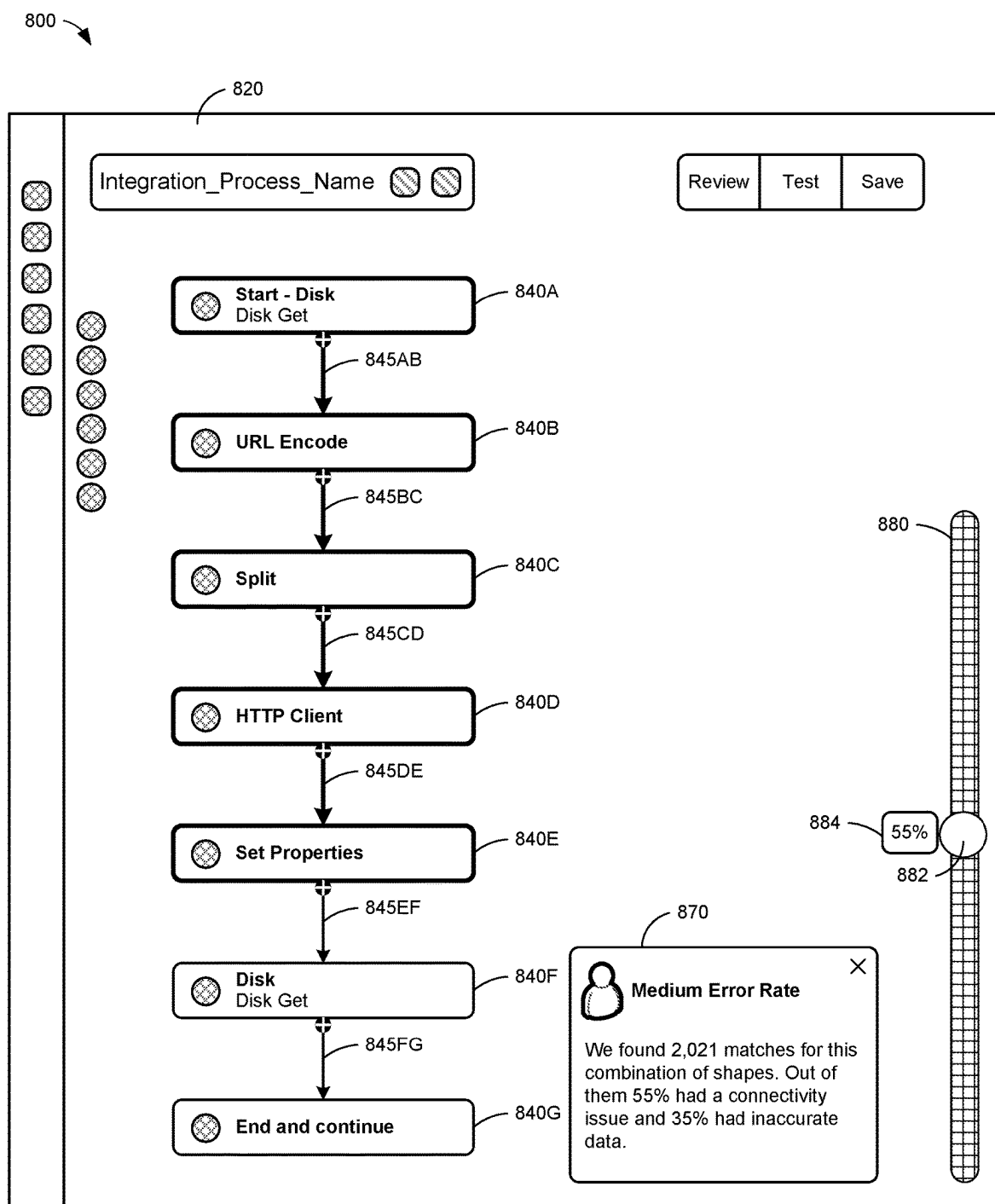


FIG. 8E

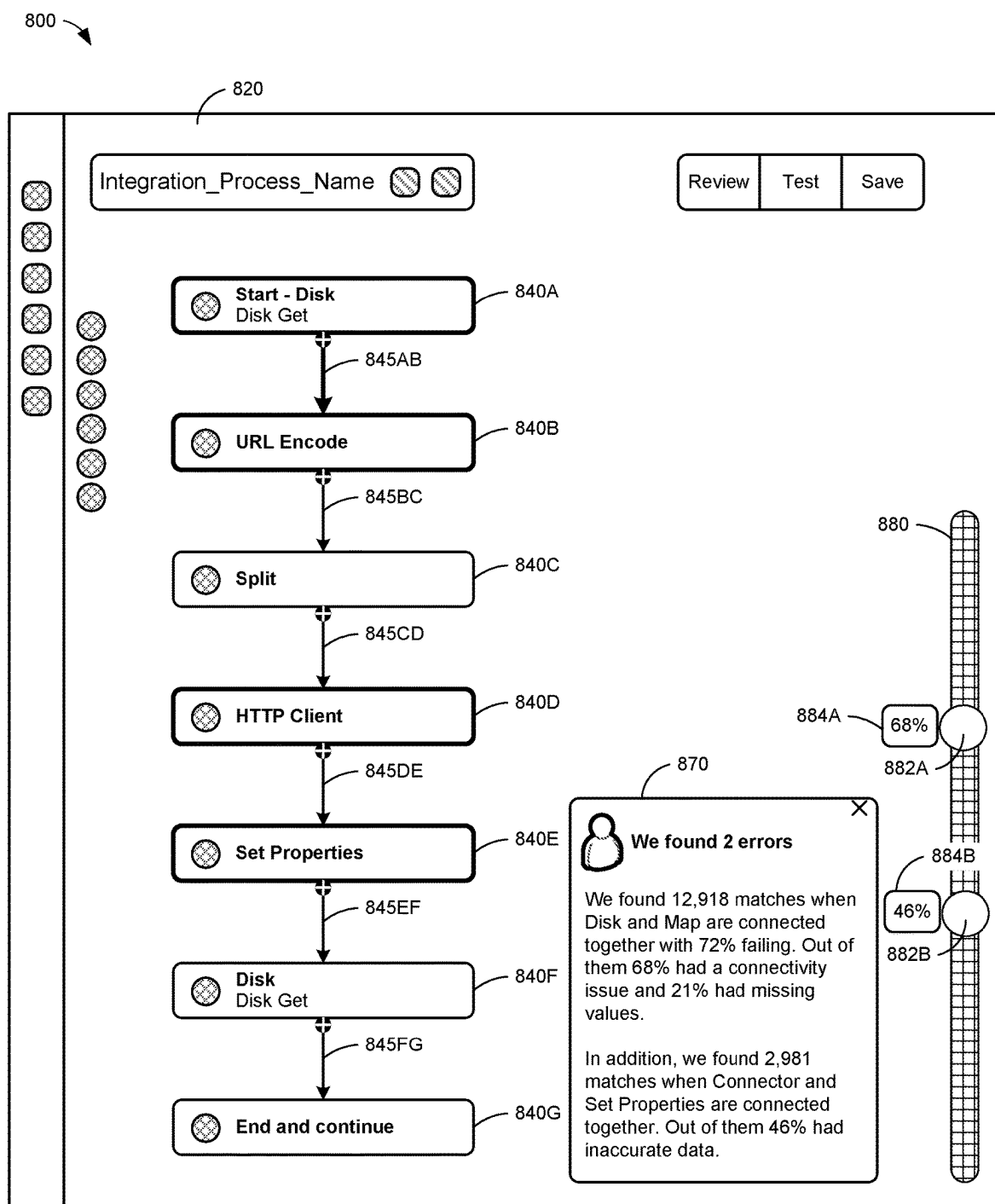


FIG. 8F

ARTIFICIAL-INTELLIGENCE-ASSISTED ERROR PREDICTION IN INTEGRATION PROCESSES

BACKGROUND

Field of the Invention

[0001] The embodiments described herein are generally directed to artificial intelligence (AI), and, more particularly, to the use of artificial intelligence to assist in the construction of integration processes by predicting errors during construction of the integration processes.

Description of the Related Art

[0002] Integration platform as a service (iPaaS) enables the integration of applications and data. The iPaaS platform provided by Boomi® of Chesterbrook, Pennsylvania, enables user to construct integration processes from pre-built steps, represented by “shapes,” which each has a set of configuration properties. Each step dictates how an integration process retrieves data, manipulates data, routes data, sends data, and/or the like. These steps can be connected together in endless combinations to build simple to very complex integration processes.

[0003] Recently, there has been a major push to simplify business processes. In particular, it is advantageous when workers can perform daily tasks without requiring special skills or training. This simplifies onboarding and empowers workers to be as efficient and productive as possible.

[0004] However, the technical knowledge required to construct integration processes is a barrier to automating the construction process. When constructing an integration process, novice users often have to go through multiple iterations of trial and error before achieving a successful, error-free implementation. In many cases, an error may not be discovered until after the integration process has been deployed, which can cause significant disruption to operations. In addition, a developer's confidence in a constructed integration process generally decreases right before deployment.

[0005] There are error detection tools available for code-heavy integration development environments. However, such tools require expertise in coding, and therefore, are not suitable for novice users. In addition, with the exception of real-time syntax checking, such tools require the user to compile and run the code, and then retroactively solve the error. In other words, the user must first write the code, and then go back and fix the errors in the written code. Currently, there is no tool for preemptively detecting errors before they become engrained in the code, let alone, in a low-code or no-code integration development environment.

SUMMARY

[0006] Accordingly, systems, methods, and non-transitory computer-readable media are disclosed to for AI-assisted construction of integration processes, to make it easier for users, including novice users, to proactively build effective, error-free integration processes. This can eliminate or otherwise mitigate operational disruptions of an organization.

[0007] In an embodiment, a method comprises using at least one hardware processor to: during a building phase, collect historical integration data from a plurality of integration platforms managed through an integration platform

as a service (iPaaS) platform, wherein the historical integration data comprise representations of a plurality of integration processes, and wherein each of the plurality of integration processes comprises at least one lineage including a sequence of steps, generate a dataset comprising representations of the lineages in the plurality of integration processes, wherein each of the representations of the lineages is associated with error information, and based on the dataset, build an error prediction model that receives a representation of a lineage as an input and produces an error prediction as an output; and during an operation phase, generate a graphical user interface comprising one or more inputs for constructing an integration process, receive a lineage including a sequence of steps from a user via the graphical user interface, and in response to a trigger, apply the error prediction model to the received lineage to produce the error prediction.

[0008] The method may further comprise using the at least one hardware processor to, in response to the trigger, further: generate a prompt using the error prediction; input the prompt to a generative language model to produce a natural-language output; and display the natural-language output in the graphical user interface. Generating the prompt may comprise inserting the error prediction into a predefined template that comprises one or both of a pre-conversation or a post-conversation.

[0009] Generating the dataset may comprise flattening each of the plurality of integration processes, comprising multiple paths through the integration process, in the historical integration data, into a plurality of lineages that each consists of a single path through the integration process. Generating the dataset may further comprise, for each of the plurality of integration processes that comprises multiple paths, including a representation of each of the plurality of lineages that consists of a single path through the integration process in the dataset. Generating the dataset may further comprise, for each of at least a subset of the plurality of integration processes, including a representation of each of one or more lineages that consist of a sub-path through the integration process in the dataset.

[0010] Each of the representations of the lineages may comprise a feature vector that includes an entry for each step in the lineage and is annotated with the error information.

[0011] The error information may comprise an execution result of a corresponding one of the plurality of integration processes. The execution result may comprise one or more errors output during execution of the corresponding integration process.

[0012] The trigger may be a user operation.

[0013] The graphical user interface may comprise a virtual canvas on which shapes, representing steps, are dragged and dropped to construct the integration process. Receiving the lineage may comprise: receiving a selection of one or more shapes on the virtual canvas; and receiving a selection of an analyze input as the trigger. Receiving the selection of one or more shapes on the virtual canvas may comprise: receiving a selection of a review input; displaying a selection box on the virtual canvas; and receiving a manipulation of the selection box by the user.

[0014] The method may further comprise using the at least one hardware processor to display at least one visual representation of the error prediction within the graphical user interface. The at least one visual representation of the error prediction may comprise a dialog that includes a description

of the error prediction. The at least one visual representation of the error prediction may comprise a severity meter that indicates a severity of the error prediction on a bar having a first end that represents least severe and a second end that represents most severe.

[0015] Each of the plurality of integration platforms may be managed by a different organizational account than one or more other ones of the plurality of integration platforms.

[0016] The method may further comprise using the at least one hardware processor to, after the building phase and prior to the operation phase, deploy the error prediction model as a microservice within the iPaaS platform.

[0017] It should be understood that any of the features in the methods above may be implemented individually or with any subset of the other features in any combination. Thus, to the extent that the appended claims would suggest particular dependencies between features, disclosed embodiments are not limited to these particular dependencies. Rather, any of the features described herein may be combined with any other feature described herein, or implemented without any one or more other features described herein, in any combination of features whatsoever. In addition, any of the methods, described above and elsewhere herein, may be embodied, individually or in any combination, in executable software modules of a processor-based system, such as a server, and/or in executable instructions stored in a non-transitory computer-readable medium.

BRIEF DESCRIPTION OF THE DRAWINGS

[0018] The details of the present invention, both as to its structure and operation, may be gleaned in part by study of the accompanying drawings, in which like reference numerals refer to like parts, and in which:

[0019] FIG. 1 illustrates an example infrastructure in which one or more of the processes described herein, may be implemented, according to an embodiment;

[0020] FIG. 2 illustrates an example processing system, by which one or more of the processes described herein may be executed, according to an embodiment;

[0021] FIG. 3 illustrates an example data flow for AI-based error prediction during construction of an integration process, according to an embodiment;

[0022] FIG. 4 illustrates a process for building an error prediction model in a building phase, according to an embodiment;

[0023] FIG. 5 illustrates a process for operating an error prediction model in an operation phase, according to an embodiment;

[0024] FIG. 6 illustrates a simple integration process 160, according to an example;

[0025] FIG. 7 illustrates a process for feature processing, according to an embodiment; and

[0026] FIGS. 8A-8F illustrate a graphical user interface, according to an embodiment.

DETAILED DESCRIPTION

[0027] In an embodiment, systems, methods, and non-transitory computer-readable media are disclosed for AI-assisted error prediction during construction of an integration process. Embodiments are intended to increase developer confidence in the construction process and reduce the learning curve in the management of an integration platform in multiple implementation scenarios, by offering

contextual assistance through the preemptive detection of errors during the construction process in a low-code integration environment. In addition, embodiments may remove the requirement of executing integration processes in order to detect errors, by preemptively alerting users about potential issues during construction of the integration processes. Embodiments may also minimize users' reliance on subject-matter experts for business applications, via recommendations provided by artificial intelligence that has been trained on data from other users and implementations of integration processes on an iPaaS platform.

[0028] After reading this description, it will become apparent to one skilled in the art how to implement the invention in various alternative embodiments and alternative applications. However, although various embodiments of the present invention will be described herein, it is understood that these embodiments are presented by way of example and illustration only, and not limitation. As such, this detailed description of various embodiments should not be construed to limit the scope or breadth of the present invention as set forth in the appended claims.

1. Example Infrastructure

[0029] FIG. 1 illustrates an example infrastructure 100, in which one or more of the processes described herein may be implemented, according to an embodiment. Infrastructure 100 may comprise a platform 110 which hosts and/or executes one or more of the disclosed processes, which may be implemented in software and/or hardware. Platform 110 may comprise dedicated servers, or may instead be implemented in a computing cloud, in which the resources of one or more servers are dynamically and elastically allocated to multiple tenants based on demand. In either case, the servers may be collocated and/or geographically distributed.

[0030] Platform 110 may be communicatively connected to one or more networks 120. Network(s) 120 enable communication between platform 110 and user system(s) 130. Network(s) 120 may comprise the Internet, and communication through network(s) 120 may utilize standard transmission protocols, such as HyperText Transfer Protocol (HTTP), HTTP Secure (HTTPS), File Transfer Protocol (FTP), FTP Secure (FTPS), Secure Shell FTP (SFTP), and the like, as well as proprietary protocols. While platform 110 is illustrated as being connected to a plurality of user systems 130 through a single set of network(s) 120, it should be understood that platform 110 may be connected to different user systems 130 via different sets of one or more networks. For example, platform 110 may be connected to a subset of user systems 130 via the Internet, but may be connected to another subset of user systems 130 via an intranet.

[0031] While only a few user systems 130 are illustrated, it should be understood that platform 110 may be communicatively connected to any number of user system(s) 130 via network(s) 120. User system(s) 130 may comprise any type or types of computing devices capable of wired and/or wireless communication, including without limitation, desktop computers, laptop computers, tablet computers, smart phones or other mobile phones, servers, game consoles, televisions, set-top boxes, electronic kiosks, point-of-sale terminals, and/or the like. However, it is generally contemplated that a user system 130 would be the personal or professional workstation of an integration developer that has a user account for accessing server application 112 on

platform 110. When platform 110 is an iPaaS platform, each user account may be associated with an overarching organizational account for managing an integration platform on the iPaaS platform.

[0032] Server application 112 may manage an integration environment 140. In particular, server application 112 may provide a user interface 150 and backend functionality, including one or more of the processes disclosed herein, to enable users, via user systems 130, to construct, develop, modify, save, delete, test, deploy, un-deploy, and/or otherwise manage integration processes 160 within integration environment 140.

[0033] The user of a user system 130 may authenticate with platform 110 using standard authentication means, to access server application 112 in accordance with permissions or roles of the associated user account. The user may then interact with server application 112 to manage one or more integration processes 160, for example, within a larger integration platform within integration environment 140. It should be understood that multiple users, on multiple user systems 130, may manage the same integration process(es) 160 and/or different integration processes 160 in this manner, according to the permissions or roles of their associated user accounts.

[0034] Although only a single integration process 160 is illustrated, it should be understood that, in reality, integration environment 140 may comprise any number of integration processes 160. In an embodiment, integration environment 140 supports integration platform as a service (iPaaS). In this case, integration environment 140 may comprise one or a plurality of integration platforms that each comprises one or more integration processes 160. Each integration platform may be associated with an organization, which may be associated with one or more user accounts by which respective user(s) manage the organization's integration platform, including the various integration process(es) 160.

[0035] An integration process 160 may represent a transaction involving the integration of data between two or more systems, and may comprise a series of elements that specify logic and transformation requirements for the data to be integrated. Each element, which may also be referred to herein as a "step" or "shape," may transform, route, and/or otherwise manipulate data to attain an end result from input data. For example, a basic integration process 160 may receive data from one or more data sources (e.g., via an application programming interface 162 of the integration process 160), manipulate the received data in a specified manner (e.g., including analyzing, normalizing, altering, updated, enhancing, and/or augmenting the received data), and send the manipulated data to one or more specified destinations. An integration process 160 may represent a business workflow or a portion of a business workflow or a transaction-level interface between two systems, and comprise, as one or more elements, software modules that process data to implement the business workflow or interface. A business workflow may comprise any myriad of workflows of which an organization may repetitively have need. For example, a business workflow may comprise, without limitation, procurement of parts or materials, manufacturing a product, selling a product, shipping a product, ordering a product, billing, managing inventory or assets, providing customer service, ensuring information security, marketing, onboarding or offboarding an employee, assessing risk, obtaining regulatory approval, reconciling data,

auditing data, providing information technology services, and/or any other workflow that an organization may implement in software.

[0036] Of particular relevance to the present disclosure, the backend functionality of server application 112 may include a process for constructing an integration process 160 within one or more screens of a graphical user interface 150. Embodiments of functionality, in server application 112, that enables the construction of integration processes 160, are disclosed, for example, in U.S. Pat. No. 8,533,661, issued on Sep. 10, 2013, and in U.S. Pat. No. 11,886,965, issued on Jan. 30, 2024, which are both hereby incorporated herein by reference as if set forth in full. In an embodiment, an AI model 114 is used to preemptively detect errors during the construction of integration process 160. The user may construct, configure, and/or finalize integration process 160 within graphical user interface 150, as facilitated by the error detection of AI model 114, and then deploy the final integration process 160 to integration environment 140. Platform 110 may also manage a database 116, which may store data used by server application 112 and/or AI model 114.

[0037] Each integration process 160, when deployed, may be communicatively coupled to network(s) 120. For example, each integration process 160 may comprise an application programming interface (API) 162 that enables clients to access integration process 160 via network(s) 120. A client may push data to integration process 160 through application programming interface 162, and/or pull data from integration process 160 through application programming interface 162.

[0038] One or more third-party systems 170 may be communicatively connected to network(s) 120, such that each third-party system 170 may communicate with an integration process 160 in integration environment 150 via application programming interface 162. Third-party system 170 may host and/or execute a software application that pushes data to integration process 160 and/or pulls data from integration process 160, via application programming interface 162. Additionally or alternatively, an integration process 160 may push data to a software application on third-party system 170 and/or pull data from a software application on third-party system 170, via an application programming interface of the third-party system 170. Thus, third-party system 170 may be a client or consumer of one or more integration processes 160, a data source for one or more integration processes 160, and/or the like. As examples, the software application on third-party system 170 may comprise, without limitation, enterprise resource planning (ERP) software, customer relationship management (CRM) software, accounting software, and/or the like.

2. Example Processing System

[0039] FIG. 2 illustrates an example processing system, by which one or more of the processes described herein may be executed, according to an embodiment. For example, system 200 may be used to store and/or execute server application 112, and/or may represent components of platform 110, user system(s) 130, third-party system 170, and/or other processing devices described herein. System 200 can be any processor-enabled device (e.g., server, personal computer, etc.) that is capable of wired or wireless data communication. Other processing systems and/or architectures may also be used, as will be clear to those skilled in the art.

[0040] System **200** may comprise one or more processors **210**. Processor(s) **210** may comprise a central processing unit (CPU). Additional processors may be provided, such as a graphics processing unit (GPU), an auxiliary processor to manage input/output, an auxiliary processor to perform floating-point mathematical operations, a special-purpose microprocessor having an architecture suitable for fast execution of signal-processing algorithms (e.g., digital-signal processor), a subordinate processor (e.g., back-end processor), an additional microprocessor or controller for dual or multiple processor systems, and/or a coprocessor. Such auxiliary processors may be discrete processors or may be integrated with a main processor **210**. Examples of processors which may be used with system **200** include, without limitation, any of the processors (e.g., Pentium™, Core i7™, Core i9™, Xeon™, etc.) available from Intel Corporation of Santa Clara, California, any of the processors available from Advanced Micro Devices, Incorporated (AMD) of Santa Clara, California, any of the processors (e.g., A series, M series, etc.) available from Apple Inc. of Cupertino, any of the processors (e.g., Exynos™) available from Samsung Electronics Co., Ltd., of Seoul, South Korea, any of the processors available from NXP Semiconductors N.V. of Eindhoven, Netherlands, and/or the like.

[0041] Processor(s) **210** may be connected to a communication bus **205**. Communication bus **205** may include a data channel for facilitating information transfer between storage and other peripheral components of system **200**. Furthermore, communication bus **205** may provide a set of signals used for communication with processor **210**, including a data bus, address bus, and/or control bus (not shown). Communication bus **205** may comprise any standard or non-standard bus architecture such as, for example, bus architectures compliant with industry standard architecture (ISA), extended industry standard architecture (EISA), Micro Channel Architecture (MCA), peripheral component interconnect (PCI) local bus, standards promulgated by the Institute of Electrical and Electronics Engineers (IEEE) including IEEE 488 general-purpose interface bus (GPIB), IEEE 696/S-100, and/or the like.

[0042] System **200** may comprise main memory **215**. Main memory **215** provides storage of instructions and data for programs executing on processor **210**, such as any of the software discussed herein. It should be understood that programs stored in the memory and executed by processor **210** may be written and/or compiled according to any suitable language, including without limitation C/C++, Java, JavaScript, Perl, Python, Visual Basic, .NET, and the like. Main memory **215** is typically semiconductor-based memory such as dynamic random access memory (DRAM) and/or static random access memory (SRAM). Other semiconductor-based memory types include, for example, synchronous dynamic random access memory (SDRAM), Rambus dynamic random access memory (RDRAM), ferroelectric random access memory (FRAM), and the like, including read only memory (ROM).

[0043] System **200** may comprise secondary memory **220**. Secondary memory **220** is a non-transitory computer-readable medium having computer-executable code and/or other data (e.g., any of the software disclosed herein) stored thereon. In this description, the term “computer-readable medium” is used to refer to any non-transitory computer-readable storage media used to provide computer-executable code and/or other data to or within system **200**. The com-

puter software stored on secondary memory **220** is read into main memory **215** for execution by processor **210**. Secondary memory **220** may include, for example, semiconductor-based memory, such as programmable read-only memory (PROM), erasable programmable read-only memory (EPROM), electrically erasable read-only memory (EEPROM), and flash memory (block-oriented memory similar to EEPROM).

[0044] Secondary memory **220** may include an internal medium **225** and/or a removable medium **230**. Internal medium **225** and removable medium **230** are read from and/or written to in any well-known manner. Internal medium **225** may comprise one or more hard disk drives, solid state drives, and/or the like. Removable storage medium **230** may be, for example, a magnetic tape drive, a compact disc (CD) drive, a digital versatile disc (DVD) drive, other optical drive, a flash memory drive, and/or the like.

[0045] System **200** may comprise an input/output (I/O) interface **235**. I/O interface **235** provides an interface between one or more components of system **200** and one or more input and/or output devices. Examples of input devices include, without limitation, sensors, keyboards, touch screens or other touch-sensitive devices, cameras, biometric sensing devices, computer mice, trackballs, pen-based pointing devices, and/or the like. Examples of output devices include, without limitation, other processing systems, cathode ray tubes (CRTs), plasma displays, light-emitting diode (LED) displays, liquid crystal displays (LCDs), printers, vacuum fluorescent displays (VFDs), surface-conduction electron-emitter displays (SEDs), field emission displays (FEDs), and/or the like. In some cases, an input and output device may be combined, such as in the case of a touch-panel display (e.g., in a smartphone, tablet computer, or other mobile device).

[0046] System **200** may comprise a communication interface **240**. Communication interface **240** allows software to be transferred between system **200** and external devices, networks, or other information sources. For example, computer-executable code and/or data may be transferred to system **200** from a network server via communication interface **240**. Examples of communication interface **240** include a built-in network adapter, network interface card (NIC), Personal Computer Memory Card International Association (PCMCIA) network card, card bus network adapter, wireless network adapter, Universal Serial Bus (USB) network adapter, modem, a wireless data card, a communications port, an infrared interface, an IEEE 1394 fire-wire, and any other device capable of interfacing system **200** with a network (e.g., network(s) **120**) or another computing device. Communication interface **240** preferably implements industry-promulgated protocol standards, such as Ethernet IEEE 802 standards, Fiber Channel, digital subscriber line (DSL), asynchronous digital subscriber line (ADSL), frame relay, asynchronous transfer mode (ATM), integrated digital services network (ISDN), personal communications services (PCS), transmission control protocol/Internet protocol (TCP/IP), serial line Internet protocol/point to point protocol (SLIP/PPP), and so on, but may also implement customized or non-standard interface protocols as well.

[0047] Software transferred via communication interface **240** is generally in the form of electrical communication signals **255**. These signals **255** may be provided to commu-

nication interface **240** via a communication channel **250** between communication interface **240** and an external system **245**. In an embodiment, communication channel **250** may be a wired or wireless network (e.g., network(s) **120**), or any variety of other communication links. Communication channel **250** carries signals **255** and can be implemented using a variety of wired or wireless communication means including wire or cable, fiber optics, conventional phone line, cellular phone link, wireless data communication link, radio frequency (“RF”) link, or infrared link, just to name a few.

[0048] Computer-executable code is stored in main memory **215** and/or secondary memory **220**. Computer-executable code can also be received from an external system **245** via communication interface **240** and stored in main memory **215** and/or secondary memory **220**. Such computer-executable code, when executed, enables system **200** to perform the various processes or functions of the disclosed embodiments.

[0049] In an embodiment that is implemented using software, the software may be stored on a computer-readable medium and initially loaded into system **200** by way of removable medium **230**, I/O interface **235**, or communication interface **240**. In such an embodiment, the software is loaded into system **200** in the form of electrical communication signals **255**. The software, when executed by processor **210**, may cause processor **210** to perform one or more of the processes or functions of the disclosed embodiments.

[0050] System **200** may optionally comprise wireless communication components that facilitate wireless communication over a voice network and/or a data network (e.g., in the case of user system **130**). The wireless communication components comprise an antenna system **270**, a radio system **265**, and a baseband system **260**. In system **200**, radio frequency (RF) signals are transmitted and received over the air by antenna system **270** under the management of radio system **265**.

[0051] In an embodiment, antenna system **270** may comprise one or more antennae and one or more multiplexors (not shown) that perform a switching function to provide antenna system **270** with transmit and receive signal paths. In the receive path, received RF signals can be coupled from a multiplexor to a low noise amplifier (not shown) that amplifies the received RF signal and sends the amplified signal to radio system **265**.

[0052] In an alternative embodiment, radio system **265** may comprise one or more radios that are configured to communicate over various frequencies. In an embodiment, radio system **265** may combine a demodulator (not shown) and modulator (not shown) in one integrated circuit (IC). The demodulator and modulator can also be separate components. In the incoming path, the demodulator strips away the RF carrier signal leaving a baseband receive audio signal, which is sent from radio system **265** to baseband system **260**.

[0053] If the received signal contains audio information, then baseband system **260** decodes the signal and converts it to an analog signal. Then, the signal is amplified and sent to a speaker. Baseband system **260** also receives analog audio signals from a microphone. These analog audio signals are converted to digital signals and encoded by baseband system **260**. Baseband system **260** also encodes the digital signals for transmission and generates a baseband transmit audio signal that is routed to the modulator portion

of radio system **265**. The modulator mixes the baseband transmit audio signal with an RF carrier signal, generating an RF transmit signal that is routed to antenna system **270** and may pass through a power amplifier (not shown). The power amplifier amplifies the RF transmit signal and routes it to antenna system **270**, where the signal is switched to the antenna port for transmission.

[0054] Baseband system **260** is communicatively coupled with processor(s) **210**, which have access to memory **215** and **220**. Thus, software can be received from baseband processor **260** and stored in main memory **210** or in secondary memory **220**, or executed upon receipt. Such software, when executed, can enable system **200** to perform the various processes or functions of the disclosed embodiments.

3. Introduction

[0055] The value of an integration platform hinges on its ability to flawlessly transfer and transform data between complex systems. However, as with any complex system, occasional errors are inevitable. These errors can result from a myriad issues, including, without limitation, coding issues, data transformation or mapping issues, network and connectivity issues, API issues, user errors, and/or the like.

[0056] Conventional error detection for integration processes **160** happens in three steps. First, the user must construct an integration process **160**. Second, the user must run the constructed integration process **160** to detect any errors. Third, the user must manually resolve the error(s) via trial and error. This three-step process relies heavily on human skills, including insight, research, creativity, and connecting diverse information to identify and resolve the underlying issues.

[0057] This conventional three-step process is inefficient for at least two reasons. Firstly, errors are only detected after deploying and operating integration process **160**. This delay in detecting errors results in significant delays in resolving the errors and incurs substantial costs in time and resource allocations. Secondly, the trial-and-error phase for resolving the errors depends heavily on the user’s understanding of integration, platform **110**, and the available data. Without significant knowledge, experience, and/or data, it will be difficult and potentially infeasible for the user to resolve the errors.

[0058] Accordingly, disclosed embodiments of server application **112** analyze the design of an integration process **160**, using AI model **114**, during the construction of integration process **160** via user interface **150**. The design of integration process **160** may be analyzed by AI model **114** in response to a user operation, in real time in the background as integration process **160** is constructed, and/or in response to another trigger. AI model **114** may comprise both an error prediction model, trained on data from historical integration data, which may comprise a massive repository of previously executed integration processes **160**, and a generative language model to summarize errors, detected by the error prediction model, in natural language. Thus, when AI model **114** detects a potential error, the user may be notified about the potential error in natural language via user interface **150**. As used herein, the term “natural language” refers to the language that would be expected from a conversation between humans. As an example, the natural-language output may indicate error-prone characteristics of integration process **160**, summarize any predicted errors, including the

percentage of integration processes 160 in which each error occurred and/or the likelihood of an error occurring, provide the reasons why such errors might occur, and/or the like. This provides the user with the ability to preemptively understand potential errors, so that the user can preemptively and efficiently troubleshoot and resolve any errors in integration process 160 before compilation and execution of integration process 160.

[0059] As used herein, the term “error” should be understood to include any execution result that may impact integration process 160. In a preferred embodiment, an execution result comprises any errors and/or warnings that are produced at compile-time and/or runtime of integration process 160. Thus, it should be understood that the term “error,” as used herein, may include both an error event that prevents integration process 160 from continuing to function and a warning event that indicates a problem from which integration process 160 is able to at least partially recover.

4. Example Data Flow

[0060] FIG. 3 illustrates an example data flow 300 for AI-based error prediction during construction of an integration process 160, according to an embodiment. In data flow 300, user interface 150 may implement modules 320 and 350, server application 112 may implement modules 315, 325, 335, and 345, AI model 114 may comprise error prediction model 330 and generative language model 340, and database 116 may store historical integration data 310. Modules 315, 320, 325, 335, 345, and 350 are preferably implemented as software modules, but could also be implemented as hardware modules or as modules comprising a combination of hardware and software.

[0061] Historical integration data 310 may comprise representations of previously constructed and executed integration processes 160, as well as any errors associated with those integration processes 160. Historical integration data 310 may be collected from a plurality of integration platforms managed and executed by an iPaaS platform, such as the Boomi® iPaaS platform. The iPaaS platform may support a plurality of integration platforms, each managed by a different organizational account that is associated with one or more user accounts. In this case, historical integration data 310 may represent a massive repository of previously executed integration processes 160 that is very diverse in terms of structures, configurations, applications, inputs and outputs, and the like, and potentially crowd-sourced from a diverse group of organizations.

[0062] Module 315, which will be described in greater detail elsewhere herein, may utilize historical integration data 310 to build (e.g., train) an error prediction model 330 of AI model 114. Error prediction model 330 is trained to accept integration data for an integration process 160 as input, and output a prediction of one or more errors, if any, associated with that integration process 160. It should be understood that the integration data that are input to error prediction model 330 may utilize the same input schema as the integration data that are used to build error prediction model 330. Notably, the massive and diverse set of previously executed integration processes 160, in historical integration data 310, enables error prediction model 330 to provide more intelligent and accurate predictions than would be possible using conventional tools.

[0063] At some subsequent time, using module 320, a user may begin constructing an integration process 160 within

user interface 150. For example, user interface 150 may comprise or consist of a graphical user interface that comprises a virtual canvas on which a user may drag and drop and connect shapes, representing steps that perform specific functions within an integration process 160. Thus, the user may intuitively construct an integration process 160 by simply placing shapes on the virtual canvas and connecting those shapes together, to define data flows between the functions represented by those shapes.

[0064] At one or more points in time, during construction of integration process 160 in module 320, an error prediction in module 325 may be executed in response to a trigger. In an embodiment, the trigger is a user operation. For instance, the user may select an analyze input within user interface 150 that triggers module 325. Alternatively or additionally, module 325 may be triggered in the background (i.e., without user involvement), in real time, in response to an event, such as any change or one or more types of changes to integration process 160, a periodic expiration of a time interval, and/or the like. For example, the trigger may be the user selecting a sequence of two or more shapes for analysis, the user adding a new shape to or removing an existing shape from integration process 160, the user reconfiguring a shape within integration process 160, the periodic expiration of each of a plurality of fixed time intervals, and/or the like. It should be understood that, as used herein, the terms “real time” and “real-time” refer to events that occur simultaneously, as well as events that are separated in time due to ordinary latencies in processing, communications, memory access, and/or the like. In any case, module 325 may be triggered, automatically (i.e., without user intervention), semi-automatically (e.g., with user confirmation), and/or manually (e.g., in response to a user operation) according to any of the above examples or in any other suitable manner.

[0065] When triggered (e.g., by user interface 150), module 325, which will be described in greater detail elsewhere herein, processes integration data, of the integration process 160 being constructed in module 320, according to an input schema for error prediction model 330, and inputs the processed integration data into error prediction model 330. Module 325 may load error prediction model 330 from database 116. Alternatively, error prediction model 330 may execute as a service that is always available to module 325 (e.g., in a microservices architecture), in which case module 325 may provide the input to error prediction model 330 via an application programming interface (API) of error prediction model 330.

[0066] The output of error prediction model 330 may be an error prediction that comprises an indication of any error that is predicted for the integration process 160 being constructed in module 320. Each indication of an error in the error prediction may be associated with a particular component (e.g., step, connection, configurable attribute, etc.) of integration process 160 or with integration process 160 as a whole. The indication may comprise an identifier of the error, a description of the error, and/or the like. The description of the error may comprise or consist of a name of the error, a summary of the error, a raw compile-time or runtime error or warning output that was associated with a historical integration process 160 in historical integration data 310, and/or the like. It should be understood that, in the event that no error is predicted by error prediction model 330, the error prediction, output by error prediction model 330, may be empty or comprise an indication that no error was predicted.

[0067] Module 335 may receive the error prediction, output by error prediction model 330, and generate a prompt for generative language model 340 using the error prediction. Module 335 may generate the prompt by inserting the error prediction, output by error prediction model 330, into a predefined template. The raw error prediction may be inserted, or the error prediction may be pre-processed before insertion. The predefined template may comprise a pre-conversation and/or post-conversation, which provide context and/or instructions for generative language model 340, and a placeholder into which the error prediction is inserted. The pre-conversation and/or post-conversation may define the role of generative language model 340 (e.g., to summarize the output of error prediction model 330), define an output format for generative language model 340 (e.g., a list structure, a hierarchical structure, a markup-language structure, etc.), and/or the like.

[0068] Module 335 may input the generated prompt to generative language model 340 of AI model 114 to produce a natural-language output. Generative language model 340 may comprise or consist of a large language model, such as the Generative Pre-trained Transformer (GPT). GPT-4 is the fourth-generation language prediction model in the GPT-n series, created by OpenAI™ of San Francisco, California. GPT-4 is an autoregressive language model that uses deep learning to produce human-like text. GPT-4 has been pre-trained on a vast amount of text from the open Internet. While GPT-4 is provided as an example, it should be understood that generative language model 340 may be any natural-language model, including past and future generations of GPT, as well as other large language models. In an embodiment, a pre-trained generative language model, such as GPT-4, is used as a base model that is fine tuned for error description and summarization to produce generative language model 340.

[0069] Module 345 may receive the natural-language output of generative language model 340 in response to the prompt. The natural-language output may comprise or consist of a summary of the error prediction, output by error prediction model 330, expressed in natural language. In the event that error prediction model 330 did not predict any errors, the natural-language output may be empty or null, or may comprise or consist of an indication (e.g., in natural language) that no errors were detected. Alternatively, in the event that error prediction model 330 did not predict any errors, execution of module 335 may be omitted, such that an indication that no errors were predicted is provided directly to module 345 without utilizing generative language model 340. When receiving a natural-language output, module 345 may process the natural-language output by formatting the natural-language output into a visual representation that is output to user interface 150. As an example, the visual representation may comprise one or more dialogs that include the natural-language output and potentially one or more inputs for interacting with the dialog. As will be discussed elsewhere herein, the visual representation may additionally or alternatively comprise a severity meter that graphically indicates a severity of the error prediction.

[0070] In module 350, user interface 150 may display the visual representation within the graphical user interface. For example, user interface 150 may be updated to display the dialog(s) and/or severity meter output by module 345. In an embodiment, each dialog and/or the severity meter is displayed as a frame overlaid on the virtual canvas that was

being used to construct integration process 160 in module 320. Thus, the user is able to see a visual representation of the predicted errors in the same screen as the constructed representation of integration process 160. Consequently, the user may easily edit integration process 160, in view of the visually represented potential error(s), to potentially resolve those error(s), for example, by removing a shape, adding a shape, replacing a shape, reconfiguring a shape (e.g., modifying an attribute of the shape), removing a connection between shapes, adding a connection between shapes, replacing a connection between shapes, and/or updating any other aspect of any component of integration process 160. If no errors were predicted by error prediction model 330, the visual representation may comprise a single dialog and/or severity meter that indicate that no errors are predicted.

[0071] In an embodiment in which user interface 150 comprises a virtual canvas and each predicted error is associated with a particular component, a visual indication of each predicted error may be displayed in association with the component with which the predicted error is associated within the virtual canvas. For example, the visual indication for a predicted error may be displayed on the virtual canvas near or in the vicinity of the component with which the predicted error is associated. The visual indication may be selectable. When a visual indication is selected, user interface 150 may responsively display the dialog for that predicted error (e.g., as a pop-up frame, expandable/collapsible frame, etc.), comprising additional information about the predicted error to which the visual indication pertains. This information may identify the predicted error, describe the predicted error, and/or the like. Alternatively, the visual indication may be a color by which the associated components are highlighted, with each predicted error being associated with a different color.

5. Building Phase

[0072] FIG. 4 illustrates a process for building an error prediction model 330, implemented by module 315, in a building phase, according to an embodiment. Module 315 may be implemented in server application 112. While module 315 is illustrated with a certain arrangement and ordering of subprocesses, module 315 may be implemented with fewer, more, or different subprocesses and a different arrangement and/or ordering of subprocesses. Furthermore, any subprocess, which does not depend on the completion of another subprocess, may be executed before, after, or in parallel with that other independent subprocess, even if the subprocesses are described or illustrated in a particular order.

[0073] The input to module 315 may be historical integration data 310. Historical integration data 310 may comprise representations of previously constructed and executed integration processes 160, as well as any execution results associated with those integration processes 160. For example, historical integration data 310 may comprise, for each of a plurality of integration processes 160, the components of that integration process 160, including the steps in that integration process 160 and the configuration of each step in that integration process 160, as well as any errors that have been generated by that integration process 160 during compilation and/or execution. In particular, historical integration data 310 may comprise, for each represented integration process 160, information about the software routine implementing that integration process 160, configurations

and settings of that integration process **160**, and execution results of that integration process **160**. The execution results may include, without limitation, execution data, execution logs, error information (e.g., for any errors and warnings), timing information, and/or the like. It should be understood that there may be many integration process **160**, represented in historical integration data **310**, that are not associated with any error.

[0074] In subprocess **410**, a dataset **415** is generated or otherwise acquired from historical integration data **310**. Dataset **415** may comprise representations of the plurality of integration processes **160** represented in historical integration data **310**. In particular, dataset **415** may comprise an annotated plurality of feature vectors, as the representations of integration processes **160**. Each feature vector may represent a single integration process **160** from historical integration data **310**, and contain the value of one or more features, derived from historical integration data **310**. In addition, each feature vector in dataset **415** may be annotated with at least the error information that is associated with the represented integration process **160** in historical integration data **310**. Additionally or alternatively, each feature vector in dataset **415** may be annotated with the timing information that is associated with the represented integration process **160** in historical integration data **310**.

[0075] Each integration process **160** will comprise at least one lineage. As used herein, the term “lineage” refers to a sequence of steps that represents a path or sub-path through the integration process **160**. An integration process **160** may contain a plurality of paths. In particular, an integration process **160** may comprise a decision step or other type of step that results in the integration process **160** branching into a plurality of paths. It should be understood that an integration process **160** may comprise multiple branches, such that there may be numerous possible paths through the integration process **160**. In an embodiment, each lineage consists of the sequence of steps in a single path or single sub-path through the integration process **160**.

[0076] In an embodiment, each of the plurality of feature vectors in dataset **415** comprises or otherwise represents a single lineage of the represented integration process **160**. If an integration process **160** comprises a plurality of lineages (i.e., paths or sub-paths), a separate feature vector may be generated for each lineage in that integration process **160**. Thus, for example, if an integration process **160** comprises three lineages, three feature vectors may be generated, annotated, and incorporated into dataset **415**. The process of generating feature vectors for each lineage in an integration process **160** may be referred to as “flattening,” which will be discussed in greater detail elsewhere herein. The integration data may also be “cleaned” when being converted into feature vectors, which will also be discussed in greater detail elsewhere herein.

[0077] In an embodiment, each of the plurality of feature vectors in dataset **415** may be annotated with error information that comprises an indication of each of one or more errors or an indication of no error. For example, if a lineage, represented by a feature vector, is associated with error information in historical integration data **310**, the feature vector may be annotated with the raw error information in historical integration data **310** or processed error information that is derived from the raw error information in historical integration data **310**. Each indication of an error may comprise a description of the error and/or an error

identifier that can be used to retrieve a description of the error. The description of the error may comprise a raw or processed compile-time or runtime error that was output by an error handler. It should be understood that, if the lineage, represented by a feature vector, is associated with a plurality of errors in historical integration data **310**, the feature vector may be annotated with error information that represents each of the plurality of errors. If the lineage, represented by a feature vector, is not associated with any errors in historical integration data **310**, the feature vector may be annotated with error information that indicates that there are no errors (e.g., a statement that no errors exist, or an error identifier representing the absence of any error).

[0078] In summary, dataset **415** may comprise representations of the lineages in the plurality of integration processes **160** in historical integration data **310**. In addition, each of these representations (e.g., feature vectors) of the lineages may be associated (e.g., annotated) with error information. Once generated, dataset **415** may be split into one or more subsets, including a building subset **416** and an evaluation subset **418**. Evaluation subset **418** could be further split into a validation subset and a testing subset. Each subset comprises or consists of a portion of dataset **415**. The division of dataset **415** into the various subsets may be performed sequentially or according to any other suitable sampling technique. Generally, building subset **416** will be significantly larger than evaluation subset **418**.

[0079] In subprocess **420**, error prediction model **330**, which receives a representation of a lineage as an input and produces an error prediction as an output, may be built, based on building subset **416**. Error prediction model **330** may be a machine-learning model or a logic-based AI model. Error prediction model **330** may comprise any suitable model, including, without limitation, an artificial neural network, such as a deep-learning neural network (DNN), recurrent neural network (RNN), graph neural network (GNN), or the like, a random forest algorithm, a linear regression algorithm, a logistic regression algorithm, a decision tree, a support vector machine (SVM), a naïve Bayes algorithm, a k-Nearest Neighbors (kNN) algorithm, a K-means algorithm, a dimensionality reduction algorithm, a gradient-boosting algorithm, a Markov chain, a compact prediction tree (CPT), or the like.

[0080] Regardless of the particular model that is used, conceptually, error prediction model **330** may match an input lineage to lineages in historical integration data **310** to determine lineages in historical integration data **310** that are identical or similar to the input lineage (e.g., via direct matching or learned inference), and output error information associated with those matching lineages in historical integration data **310**. For example, an input lineage of A-B-C may essentially be matched to lineages of A-B-C in historical integration data **310**, and error prediction model **330** may return the execution result(s) (e.g., error and warning events) for the matched lineages in historical integration data **310**.

[0081] In an embodiment in which error prediction model **330** is a machine-learning model, subprocess **420** may comprise training error prediction model **330**, using supervised learning, to predict error information, given a feature vector that represents a lineage as input. In particular, error prediction model **330** may be trained by minimizing a loss function over a plurality of training iterations. In each training iteration, one feature vector from building subset **416** may be input to error prediction model **330** to output

predicted error information, the loss function may calculate an error between the predicted error information and the error information with which the feature vector is annotated, and one or more weights in error prediction model 330 may be adjusted, according to a suitable technique (e.g., gradient descent), to reduce the error. A training iteration may be performed for each of the annotated plurality of feature vectors in building subset 416.

[0082] In an alternative embodiment in which error prediction model 330 is a machine-learning model, error prediction model 330 may be trained using unsupervised learning. In this case, lineages, represented as feature vectors in building subset 416, may be clustered using any suitable clustering technique. Each feature vector may still be annotated with error information, such that the error information may be retrieved for any given feature vector or cluster of feature vectors during operation of error prediction model 330.

[0083] In subprocess 430, error prediction model 330, built in subprocess 420, may be evaluated. The evaluation may comprise validating and/or testing error prediction model 330 using evaluation subset 418 of dataset 415. The result of subprocess 430 may be a performance measure for error prediction model 330, such as an accuracy of error prediction model 330. Any suitable evaluation method and performance measure may be used.

[0084] In subprocess 440, it is determined whether or not error prediction model 330, built in subprocess 420, is acceptable based on the evaluation performed in subprocess 430. For example, the performance measure from subprocess 430 may be compared to a threshold or one or more other criteria. If the performance measure satisfies the criteria (e.g., is greater than or equal to the threshold), error prediction model 330 may be determined to be acceptable (i.e., “Yes” in subprocess 440). Conversely, if the performance measure does not satisfy the criteria (e.g., is less than the threshold), error prediction model 330 may be determined to be unacceptable (i.e., “No” in subprocess 440). When error prediction model 330 is determined to be acceptable (i.e., “Yes” in subprocess 440), module 315 may proceed to subprocess 450. Otherwise, when error prediction model 330 is determined to be unacceptable (i.e., “No” in subprocess 440), module 315 may return to subprocess 410 to rebuild or modify error prediction model 330 (e.g., using a new or modified dataset 415).

[0085] In subprocess 450, error prediction model 330 may be deployed. In particular, after the building phase and prior to the operation phase, error prediction model 330 may be deployed by moving error prediction model 330 from a development environment to a production environment of platform 110. For example, error prediction model 330 may be deployed as a microservice that is available at an address on platform 110 (e.g., an iPaaS platform) that is accessible to server application 112. Alternatively, error prediction model 330 may be comprised in server application 112.

6. Operation Phase

[0086] FIG. 5 illustrates a process for operating error prediction model 330, implemented by module 325, in an operation phase, according to an embodiment. Module 325 may be implemented in server application 112. While module 325 is illustrated with a certain arrangement and ordering of subprocesses, module 325 may be implemented with fewer, more, or different subprocesses and a different

arrangement and/or ordering of subprocesses. Furthermore, any subprocess, which does not depend on the completion of another subprocess, may be executed before, after, or in parallel with that other independent subprocess, even if the subprocesses are described or illustrated in a particular order.

[0087] Initially, in subprocess 510, integration data are received. The integration data may represent an integration process 160 under construction in module 320. In particular, when module 325 is triggered, the integration data, representing the integration process 160 under construction, may be provided to module 325. This integration data may comprise the components of integration process 160, including the steps in integration process 160 and the configuration of each step in integration process 160. In particular, the integration data may comprise, for the integration process 160 under construction, information about the software routine implementing the integration process 160, and configurations and settings of the integration process 160.

[0088] In subprocess 520, features may be extracted from the integration data received in subprocess 510. In particular, at least one feature vector may be generated from the integration data in the same or similar manner as feature vectors were extracted from historical integration data 310 in subprocess 410 of module 315, and according to the same input schema of error prediction model 330. Consequently, the feature vector may represent a lineage of the integration process 160 under construction. In an embodiment, a separate feature vector may be generated for each lineage (e.g., path and/or sub-path) in the integration process 160 under construction. When processing the integration data into their corresponding feature vectors, subprocess 520 may utilize the same flattening and cleaning as in subprocess 410. However, it should be understood that the resulting feature vector will not be annotated, in contrast to the annotated feature vectors in dataset 415.

[0089] In subprocess 530, error prediction model 330, which was trained in subprocess 420 of module 315 and deployed by subprocess 450 of module 315, may be applied to the features, extracted in subprocess 520. In particular, each feature vector, generated in subprocess 520, may be input to error prediction model 330. If there are multiple feature vectors (i.e., representing multiple lineages), each feature vector may be input to error prediction model 330, individually in serial or in parallel, or collectively as a batch. When applied to a feature vector, error prediction model 330 may predict zero, one, or more errors as an error prediction for the lineage of integration process 160 represented by the feature vector.

[0090] The error prediction may comprise an indication of each error. As mentioned elsewhere herein, the term “error” may refer to any execution result, including both errors and warnings. Each indication of an error may comprise a description of the error and/or an error identifier that can be used to retrieve a description of the error. In an embodiment in which error prediction model 330 is built using timing information, the error prediction may also comprise predicted timing information.

[0091] As discussed above, each feature vector may represent a lineage, including a sequence of steps, specified or otherwise received by a user during construction of an integration process 160 in module 320. In this case, error prediction model 330 is applied to this lineage to produce the error prediction. When an integration process 160 is flat-

tened into a plurality of lineages in subprocess 520, error prediction model 330 may be applied to each individual one of the feature vectors, representing the plurality of lineages, serially or in parallel. Then, the individual error predictions, output by error prediction model 330 for each of the feature vectors, may be combined into a single error prediction.

[0092] In subprocess 540, the error prediction, output by error prediction model 330 and comprising or consisting of the predicted error(s), if any, for the integration process 160 under construction, may be input to one or more downstream functions. For example, the error prediction may be provided as input to module 335, which may generate a prompt for generative language model 340 based on the predicted error(s).

7. Feature Processing

[0093] FIG. 6 illustrates a simple integration process 160, according to an example. As illustrated, this exemplary integration process 160 comprises steps A, B, C, D, E, F, G, and H, with step A representing a starting step, step F representing an ending step, and steps C and D representing branching (e.g., decision) steps.

[0094] FIG. 7 illustrates a process 700 for feature processing, according to an embodiment. Process 700 will be described with reference to the exemplary integration process 160 illustrated in FIG. 6. Process 700 may be implemented in subprocess 410 of module 315 of server application 112 and/or subprocess 520 of module 325 of server application 112 to generate the described feature vectors for the lineages. While process 700 is illustrated with a certain arrangement and ordering of subprocesses, process 700 may be implemented with fewer, more, or different subprocesses and a different arrangement and/or ordering of subprocesses. Furthermore, any subprocess, which does not depend on the completion of another subprocess, may be executed before, after, or in parallel with that other independent subprocess, even if the subprocesses are described or illustrated in a particular order.

[0095] In subprocess 710, integration data, representing an integration process 160, are acquired. It should be understood that in the case of feature processing in subprocess 410, the integration data will be acquired from historical integration data 310, whereas, in the case of feature processing in subprocess 520, the integration data will be acquired from module 320 for an integration process 160 under construction. In either case, the integration data may comprise the components of integration process 160, including the steps in integration process 160 and the configuration of each step in integration process 160. For simplicity, it will be assumed that process 700 acts on integration data for a single integration process 160. However, it should be understood that, in practice, process 700 may act on batches of integration data for a plurality of integration processes 160.

[0096] The integration data may comprise a representation of the sequence of steps in the represented integration process 160, for example, as a sequence of step identifiers. In addition, integration process 160 and/or each step in integration process 160 may be associated in the integration data with metadata. The metadata may comprise the configuration properties of integration process 160 and/or each step in integration process 160. For example, the configuration properties of a step may include, without limitation, an identifier of the prior step, a type of the prior step, an identifier of the next step, a type of the next step, a type of

connector represented by the step, a type of action configured for the step (e.g., get, send, execute, upsert, etc.), an end point of the connector represented by the step, an object of the connector represented by the step, the input(s) to the step, the type of input to the step, the output(s) from the step, the type of output from the step, and/or the like. Notably, not all steps in an integration process 160 will necessarily have values for every metadata field. Additionally or alternatively, the metadata may comprise statistical properties of integration process 160 and/or each step in integration process 160.

[0097] In subprocess 720, the integration data, acquired in subprocess 710, are flattened. During this flattening, an integration process 160 that comprises a plurality of paths may be flattened into a plurality of lineages, in which each lineage consists of a single one of the plurality of paths. For example, an integration process 160 that comprises multiple branching paths may be divided into a plurality of lineages that each consists of a single path from a starting step to an ending step of the integration process 160. It should be understood that an integration process 160 that consists of a single path would not need to be flattened in this manner.

[0098] The exemplary integration process 160 would be flattened into at least three lineages in subprocess 720:

[0099] (1) A→B→C→D→E→F;

[0100] (2) A→B→C→E→F; and

[0101] (3) A→B→C→D→G→H→F.

[0102] In an embodiment, each path may comprise both a starting step (e.g., A in this example) and an ending step (e.g., F in this example) of the integration process 160 from which it is derived. In other words, subprocess 720 may flatten each integration process 160, comprising multiple paths through integration process 160, whether in historical integration data 310 or in the integration data of an integration process 160 under construction, into a plurality of lineages that each consists of a single path through integration process 160. It should be understood that, in the case of subprocess 410, a representation of each of this plurality of lineages may be included in dataset 415. In the case of subprocess 520, a representation of each of this plurality of lineages may be input to error prediction model 330.

[0103] In an embodiment, in addition to lineages with starting and ending steps, an integration process 160 may be flattened into lineages that each consists of a sub-path between a starting and ending step. For example, in addition to lineages (1)-(3) above, the exemplary integration process 160 could also produce sub-lineages, including, for example, A→B, A→B→C, A→B→C→D, A→B→C→D→E, A→B→C→E, A→B→C→D→G, A→B→C→D→G→H, B→C, B→C→D, B→C→D→E, B→C→D→E→F, C→E, C→E→F, C→D, C→D→G, and so on and so forth. More generally, subprocess 720 could flatten an integration process 160, whether consisting of a single path or comprising multiple paths, into any set of paths (i.e., lineages) or sub-paths (i.e., sub-lineages) that comprise two or more steps. In other words, subprocess 720 may flatten an integration process 160 into one or more lineages that each consists of a sub-path through integration process 160. In the case of subprocess 410, a representation of each of these lineage(s) may be included in dataset 415. In the case of subprocess 520, a representation of each of these lineage(s) may be input to error prediction model 330. For the sake of simplicity, the term “lineage” should be understood to refer to both the complete lineage (i.e., consisting of a full path from

the start step to an end step) of an integration process 160 and a sub-lineage (i.e., consisting of a sub-path) of an integration process 160.

[0104] In subprocess 730, an exploratory data analysis may be performed on the flattened integration data, comprising representations of the lineages and their associated metadata (e.g., configuration properties, statistical properties, etc.). Exploratory data analysis may comprise analyzing a distribution of the integration data (e.g., as histograms, bar plots, etc.), identifying patterns, checking for missing values, identifying imbalances, identifying outliers, identifying duplicate data, identifying essential variables, identifying non-essential variables, checking assumptions, and/or the like. In general, exploratory data analysis may be used to determine how to clean a large set of data, such as historical integration data 310. Such exploratory data analysis may not be relevant to module 325, which will generally process only a single integration process 160. Thus, subprocess 730 may be omitted from subprocess 520 of module 325.

[0105] In subprocess 740, the flattened data are cleaned to produce a clean feature vector that conforms to the input schema of error prediction model 330. If subprocess 730 was executed, the cleaning may be based on the results of the exploratory data analysis in subprocess 730. In this case, cleaning may include, without limitation, removing lineages with missing values, removing lineages with outlying values, removing duplicate data, removing non-essential variables, fixing imbalances (e.g., by discarding data and/or synthesizing data), and/or the like. In an embodiment, cleaning comprises removing duplicate lineages, removing lineages with less than a certain number of steps (e.g., less than two steps), removing lineages with multiple start steps, removing connector identifiers for connector steps and/or positional arguments for other types of steps, normalizing the remaining data, and/or the like. In an embodiment, module 325 may receive integration data that are already clean (e.g., module 320 may act as a gatekeeper against noisy data), in which case subprocess 740 may be omitted from subprocess 520 of module 325.

[0106] In an embodiment, each lineage, resulting from the flattening in subprocess 720 and/or the cleaning in subprocess 740, is converted into a feature vector. Each feature vector may comprise an entry for each step in the lineage. Each entry may comprise at least the step identifier for the respective step. Each entry may also comprise one or more configuration properties, statistical properties, or other metadata associated with the respective step. In this case, each entry may be a tuple of values representing one of the steps in the lineage. The entries may be ordered, within the feature vector, in the order that the respective steps appear in the represented lineage. For example, for lineage (1) from the exemplary integration process 160, the feature vector may be represented as

$$F_1 = ([A, P_{A1}, P_{A2}, \dots, P_{AN}], [B, P_{B1}, P_{B2}, \dots, P_{BN}], \dots, [F, P_{F1}, P_{F2}, \dots, P_{FN}])$$

in which p_{ij} represents the value of a property j in the metadata for step i . In any case, the feature vector may be output by the feature engineering of process 700 to a downstream function, such as subprocess 420 in module 315 or subprocess 530 in module 325.

[0107] In the case of subprocess 410, each feature vector may be annotated with error information. The error information may comprise an execution result of the corresponding integration process 160 that produced the lineage rep-

resented by the feature vector. As discussed elsewhere herein, the execution result may comprise one or more errors (e.g., including error events and/or warning events) that were output during execution of the corresponding integration process 160.

8. Graphical User Interface

[0108] Boomi® provides an iPaaS platform that has revolutionized the integration/middleware space with a drag-and-drop graphical user interface that eliminates the need for custom code in the construction of integration processes 160. In particular, the graphical user interface comprises a virtual canvas over which a user may drag and drop shapes, representing steps that perform specific functions, and connect the shapes to define data flows between their respective functions. Thus, the user may intuitively construct an integration process 160 by simply adding, configuring, and connecting shapes in an intuitive manner.

[0109] However, prior to deployment, developers are often uncertain about whether or not the integration processes 160 that they construct will actually run. Accordingly, disclosed embodiments provide an easy-to-use, intuitive graphical user interface for predicting errors in integration processes 160 under construction. This graphical user interface may be used by both novice and expert developers to efficiently troubleshoot their integration processes 160 prior to deployment. An embodiment of this graphical user interface is described below.

[0110] FIG. 8A illustrates an example graphical user interface 800 that may be used to construct an integration process 160, according to an embodiment. Graphical user interface 800 may be provided by user interface 150 of server application 112. In the illustrated example, graphical user interface 800 comprises a navigation bar 810 and a virtual canvas 820. Virtual canvas 820 enables a user to drag and drop representations (i.e., “shapes”) of steps at positions within an integration process 160 to be constructed, and connect these representations to form one or more lineages (i.e., paths or sub-paths).

[0111] Virtual canvas 820 may comprise a shape palette 822, from which new shapes can be dragged and dropped on virtual canvas 820, and a header 824 which may comprise information (e.g., name) for the integration process 160 as a whole. In addition, virtual canvas 820 may comprise a review input 832 for triggering the disclosed error prediction for integration process 160, a test input 834 for testing integration process 160 (e.g., executing integration process 160 in a test environment), and a save input 836 for saving integration process 160 in the current configuration.

[0112] In the illustrated example, a user has constructed an integration process 160 with shapes 840A, 840B, 840C, 840D, 840E, 840F, and 840G, which each represents a step in integration process 160. Each of shapes 840 is connected to at least one adjacent shape 840 by a connection 845. In the illustrated example, shape 840A is connected to shape 840B by connection 845AB, shape 840B is connected to shape 840C by connection 845BC, shape 840C is connected to shape 840D by connection 845CD, shape 840D is connected to shape 840E by connection 845DE, shape 840E is connected to shape 840F by connection 845EF, and shape 840F is connected to shape 840G by connection 845FG. Since there are no branches, integration process 160 consists of a single path, and therefore, a single start-to-end lineage.

However, if sub-paths are considered, integration process 160 comprises a plurality of other lineages representing sub-paths.

[0113] FIG. 8B illustrates graphical user interface 800, after a user has selected review input 832, according to an embodiment. Responsively, graphical user interface 800 has been updated to display a selection box 850 with corner points 855, and a prompt dialog 860 with an analyze input 865, on virtual canvas 820. Prompt dialog 860 prompts the user to choose a review area. The user may select and drag any of corner points 855 to resize and reshape selection box 850. In this manner, the user may select one or more shapes 840 representing the precise lineage that the user wishes to analyze. For instance, in the illustrated example, the user has manipulated selection box 850 to select a lineage, represented by shapes 840A-840E, and excluding shapes 840F and 840G. From the perspective of server application 112, server application 112 may receive the selection of one or more shapes 840, representing a lineage, on virtual canvas 820 by receiving a selection of review input 832, displaying selection box 850 on virtual canvas 820, and receiving a manipulation of selection box 850 by the user.

[0114] The user may then select analyze input 865 to trigger the error prediction. In particular, selection of analyze input 865 may trigger module 325. In this case, graphical user interface 800 receives a lineage, consisting of the sequence of steps represented by the shapes 840 in selection box 850, from the user, and passes integration data for this lineage to module 325. As discussed elsewhere herein, module 325 will apply error prediction model 330 to this lineage, and potentially sub-lineages, as represented by one or more feature vector(s), to produce an error prediction. Module 335 may generate a prompt using the error prediction, and input that prompt to generative language model 340 to produce a natural-language output. Module 345 may process the natural-language output into a dialog that is displayed on graphical user interface 800 by module 350.

[0115] FIGS. 8C-8F illustrates graphical user interface 800, after the user has selected analyze input 865, according to an embodiment. In particular, FIG. 8C represents graphical user interface 800 when no errors are predicted, FIG. 8D represents graphical user interface 800 when a severe error is predicted, FIG. 8E represents graphical user interface 800 when a medium error is predicted, and FIG. 8F represents graphical user interface 800 when a plurality of errors are predicted for distinct sub-lineages.

[0116] In each case, the set of shapes 840 and connections 845, which were within selection box 850 when analyze input 865 was selected and are impacted by a predicted error, may be highlighted within virtual canvas 820. The highlighting may be color-coded, with the color of the highlighting selected based on the severity of the predicted error associated with those shapes 840 and/or connections 845. For example, the color green may be selected in the absence of a predicted error, the color yellow or orange may be selected for errors of low or medium severity, and the color red may be selected for errors of high severity.

[0117] In addition, in each case, graphical user interface 800 has been updated to include dialog 870. It should be understood that dialog 870 corresponds to the dialog that is output by module 345. Dialog 870 comprises a description of the error prediction. This description may comprise a natural-language expression that indicates the predicted error(s), if any. It should be understood that the natural-

language expression may represent the natural-language output of generative language model 340.

[0118] In addition, in each case, graphical user interface 800 may be updated to include a severity meter 880. Severity meter 880 may comprise and indicate severity on a bar with a first end (e.g., lower end) representing least severe (i.e., no errors) and a second end (e.g., upper end) representing most severe. The bar may be shaded according to a color spectrum, for example, from green at the first end to red at the second end, to indicate the progression of severity. In addition, severity meter 880 may comprise a marker 882 and a severity indication 884 that are positioned at a location, with respect to the bar, that is indicative of the severity of the represented error. For example, marker 882 and severity indication 884 may be located at the first end when the severity is 0% (i.e., no errors), located at the second end when the severity is 100%, located 25% of the length between the first end and the second end when the severity is 25%, located 50% of the length between the first end and the second end when the severity is 50%, and so on and so forth. The color-coding of the highlighting of shapes 840 and/or connections 845 may correspond to the color at the particular location, in the color spectrum in the bar of severity meter 880, at which marker 882 and severity indication 884 are positioned.

[0119] In FIG. 8C, there were no errors predicted by error prediction model 330. Thus, dialog 870 expresses the absence of errors, and severity meter 880 indicates a severity of 0%. The selected shapes 840A-840E and selected connections 845AB-845DE may be highlighted in the color of severity meter 880 at the location of marker 882. In this case, the color may be green to indicate the absence of any error.

[0120] In FIG. 8D, there was a high error rate detected. In particular, the error prediction indicated that 75% of similar lineages had at least one error. Dialog 870 expresses information about the predicted errors, including the percentage of similar lineages having each of a plurality of errors (e.g., a connectivity issue, missing values, and inaccurate data). Severity meter 880 indicates a severity of 75%, representing the percentage of similar lineages having at least one error. The selected shapes 840A-840E and selected connections 845AB-845DE may be highlighted in the color of severity meter 880 at the location of marker 882.

[0121] In FIG. 8E, there was a medium error rate detected. In particular, the error prediction indicated that 55% of similar lineages had at least one error. Dialog 870 expresses information about the predicted errors, including the percentage of similar lineages having each of a plurality of errors (e.g., a connectivity issue, and inaccurate data). Severity meter 880 indicates a severity of 55%, representing the percentage of similar lineages having at least one error. The selected shapes 840A-840E and selected connections 845AB-845DE may be highlighted in the color of severity meter 880 at the location of marker 882.

[0122] In FIG. 8F, error rates of differing severities were detected for two discrete sub-lineages. In particular, the error prediction indicated that 68% of lineages similar to a first sub-lineage, represented by shapes 840A-840B and their connection 845AB, had at least one error, and 46% of lineages similar to a second sub-lineage, represented by shapes 840D-840E and their connection 845DE, had at least one error. Dialog 870 expresses information about the predicted errors for each sub-lineage, including the percentage of similar lineages having each of one or more errors.

Severity meter **880** indicates a severity of 68% for the first sub-lineage via marker **882A** and severity indication **884A**, and a severity of 46% for the second sub-lineage via marker **882B** and severity indication **884B**. The first sub-lineage may be highlighted in the color of severity meter **880** at the location of marker **882A**, and the second sub-lineage may be highlighted in the color of severity meter **880** at the location of marker **882B**.

[0123] The above description of the disclosed embodiments is provided to enable any person skilled in the art to make or use the invention. Various modifications to these embodiments will be readily apparent to those skilled in the art, and the general principles described herein can be applied to other embodiments without departing from the spirit or scope of the invention. Thus, it is to be understood that the description and drawings presented herein represent a presently preferred embodiment of the invention and are therefore representative of the subject matter which is broadly contemplated by the present invention. It is further understood that the scope of the present invention fully encompasses other embodiments that may become obvious to those skilled in the art and that the scope of the present invention is accordingly not limited.

[0124] As used herein, the terms “comprising,” “comprise,” and “comprises” are open-ended. For instance, “A comprises B” means that A may include either: (i) only B; or (ii) B in combination with one or a plurality, and potentially any number, of other components. In contrast, the terms “consisting of,” “consist of,” and “consists of” are closed-ended. For instance, “A consists of B” means that A only includes B with no other component in the same context.

[0125] Combinations, described herein, such as “at least one of A, B, or C,” “one or more of A, B, or C,” “at least one of A, B, and C,” “one or more of A, B, and C,” and “A, B, C, or any combination thereof” include any combination of A, B, and/or C, and may include multiples of A, multiples of B, or multiples of C. Specifically, combinations such as “at least one of A, B, or C,” “one or more of A, B, or C,” “at least one of A, B, and C,” “one or more of A, B, and C,” and “A, B, C, or any combination thereof” may be A only, B only, C only, A and B, A and C, B and C, or A and B and C, and any such combination may contain one or more members of its constituents A, B, and/or C. For example, a combination of A and B may comprise one A and multiple B’s, multiple A’s and one B, or multiple A’s and multiple B’s.

What is claimed is:

1. A method comprising using at least one hardware processor to:

during a building phase,

collect historical integration data from a plurality of integration platforms managed through an integration platform as a service (iPaaS) platform, wherein the historical integration data comprise representations of a plurality of integration processes, and wherein each of the plurality of integration processes comprises at least one lineage including a sequence of steps,

generate a dataset comprising representations of the lineages in the plurality of integration processes, wherein each of the representations of the lineages is associated with error information, and

based on the dataset, build an error prediction model that receives a representation of a lineage as an input and produces an error prediction as an output; and during an operation phase,

generate a graphical user interface comprising one or more inputs for constructing an integration process, receive a lineage including a sequence of steps from a user via the graphical user interface, and

in response to a trigger, apply the error prediction model to the received lineage to produce the error prediction.

2. The method of claim 1, further comprising using the at least one hardware processor to, in response to the trigger, further:

generate a prompt using the error prediction;

input the prompt to a generative language model to produce a natural-language output; and

display the natural-language output in the graphical user interface.

3. The method of claim 2, wherein generating the prompt comprises inserting the error prediction into a predefined template that comprises one or both of a pre-conversation or a post-conversation.

4. The method of claim 1, wherein generating the dataset comprises flattening each of the plurality of integration processes, comprising multiple paths through the integration process, in the historical integration data, into a plurality of lineages that each consists of a single path through the integration process.

5. The method of claim 4, wherein generating the dataset further comprises, for each of the plurality of integration processes that comprises multiple paths, including a representation of each of the plurality of lineages that consists of a single path through the integration process in the dataset.

6. The method of claim 5, wherein generating the dataset further comprises, for each of at least a subset of the plurality of integration processes, including a representation of each of one or more lineages that consist of a sub-path through the integration process in the dataset.

7. The method of claim 1, wherein each of the representations of the lineages comprises a feature vector that includes an entry for each step in the lineage and is annotated with the error information.

8. The method of claim 1, wherein the error information comprises an execution result of a corresponding one of the plurality of integration processes.

9. The method of claim 8, wherein the execution result comprises one or more errors output during execution of the corresponding integration process.

10. The method of claim 1, wherein the trigger is a user operation.

11. The method of claim 1, wherein the graphical user interface comprises a virtual canvas on which shapes, representing steps, are dragged and dropped to construct the integration process.

12. The method of claim 11, wherein receiving the lineage comprises:

receiving a selection of one or more shapes on the virtual canvas; and

receiving a selection of an analyze input as the trigger.

13. The method of claim 12, wherein receiving the selection of one or more shapes on the virtual canvas comprises:

receiving a selection of a review input;
displaying a selection box on the virtual canvas; and
receiving a manipulation of the selection box by the user.

14. The method of claim **1**, further comprising using the at least one hardware processor to display at least one visual representation of the error prediction within the graphical user interface.

15. The method of claim **14**, wherein the at least one visual representation of the error prediction comprises a dialog that includes a description of the error prediction.

16. The method of claim **14**, wherein the at least one visual representation of the error prediction comprises a severity meter that indicates a severity of the error prediction on a bar having a first end that represents least severe and a second end that represents most severe.

17. The method of claim **1**, wherein each of the plurality of integration platforms is managed by a different organizational account than one or more other ones of the plurality of integration platforms.

18. The method of claim **1**, further comprising using the at least one hardware processor to, after the building phase and prior to the operation phase, deploy the error prediction model as a microservice within the iPaaS platform.

19. A system comprising:

at least one hardware processor; and
software that is configured to, when executed by the at least one hardware processor,
during a building phase,

collect historical integration data from a plurality of integration platforms managed through an integration platform as a service (iPaaS) platform, wherein the historical integration data comprise representations of a plurality of integration processes, and wherein each of the plurality of integration processes comprises at least one lineage including a sequence of steps,

generate a dataset comprising representations of the lineages in the plurality of integration processes, wherein each of the representations of the lineages is associated with error information, and

based on the dataset, build an error prediction model that receives a representation of a lineage as an input and produces an error prediction as an output, and

during an operation phase,

generate a graphical user interface comprising one or more inputs for constructing an integration process,

receive a lineage including a sequence of steps from a user via the graphical user interface, and

in response to a trigger, apply the error prediction model to the received lineage to produce the error prediction.

20. A non-transitory computer-readable medium having instructions stored therein, wherein the instructions, when executed by a processor, cause the processor to:

during a building phase,

collect historical integration data from a plurality of integration platforms managed through an integration platform as a service (iPaaS) platform, wherein the historical integration data comprise representations of a plurality of integration processes, and wherein each of the plurality of integration processes comprises at least one lineage including a sequence of steps,

generate a dataset comprising representations of the lineages in the plurality of integration processes, wherein each of the representations of the lineages is associated with error information, and

based on the dataset, build an error prediction model that receives a representation of a lineage as an input and produces an error prediction as an output; and

during an operation phase,

generate a graphical user interface comprising one or more inputs for constructing an integration process, receive a lineage including a sequence of steps from a user via the graphical user interface, and

in response to a trigger, apply the error prediction model to the received lineage to produce the error prediction.

* * * * *