US012393174B2

(12) **United States Patent**
Timisescu et al.

(10) **Patent No.: US 12,393,174 B2**
(45) **Date of Patent: Aug. 19, 2025**

(54) **TYPE CACHE FOR PACKAGE MANAGEMENT OF ROBOTIC PROCESS AUTOMATIONS FIELD**

(71) Applicant: **UIPATH, INC.**, New York, NY (US)

(72) Inventors: **Andrei-Sebastian Timisescu**, Bucharest (RO); **Adrian Nicolae Ignat**, London (GB); **Petrisor Cosmin Sandu**, Afumati (RO); **Bogdan Alexandru Toma**, Bucharest (RO)

(73) Assignee: **UiPath, Inc.**, New York, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 545 days.

(21) Appl. No.: **17/885,132**

(22) Filed: **Aug. 10, 2022**

(65) **Prior Publication Data**

US 2024/0053727 A1 Feb. 15, 2024

(51) **Int. Cl.**
G05B 19/4155 (2006.01)
G06F 12/0802 (2016.01)

(52) **U.S. Cl.**
CPC ***G05B 19/4155*** (2013.01); ***G06F 12/0802*** (2013.01); *G05B 2219/50391* (2013.01); *G06F 2212/60* (2013.01)

(58) **Field of Classification Search**
CPC G05B 19/4155; G05B 2219/50391; G06F 12/0802; G06F 2212/60
USPC 717/120–123, 170–178
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

7,555,755	B2 *	6/2009	Fairweather	G06F 9/4493 719/321
8,707,293	B2 *	4/2014	Festi	G06F 8/61 717/176
8,719,339	B2 *	5/2014	Reisman	G06Q 20/20 705/344
12,159,101	B1 *	12/2024	Petrescu	G06F 9/451
2020/0004798	A1 *	1/2020	Weinert, Jr.	G06F 16/9577
2023/0191601	A1 *	6/2023	Grigore	B25J 9/1661 700/246

OTHER PUBLICATIONS

Spinellis, "Package Management Systems", 2012, IEEE, pp. 84-86. (Year: 2012).*
Smita Parte et. al., "A Strategy for Package Transformation for Linux Environment", 2012, International Journal of Computer Technology and Applications, pp. 1418-1421. (Year: 2012).*

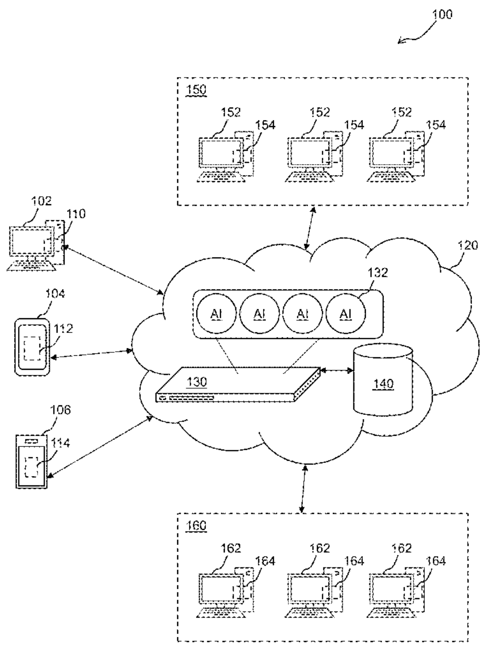
* cited by examiner

Primary Examiner — Ted T. Vo
(74) *Attorney, Agent, or Firm* — Volpe Koenig

(57) **ABSTRACT**

According to one or more embodiments, a method executed by a type cache service implemented as a computer program within a computing environment is provided. The method includes scanning feeds publishing unpacked packages and automatically detecting new activities on the feeds. The method includes indexing the new activities and all previously detected activities according to type and generating a type cache for the unpacked packages according to the indexing of the new activities and all previously detected activities.

20 Claims, 12 Drawing Sheets



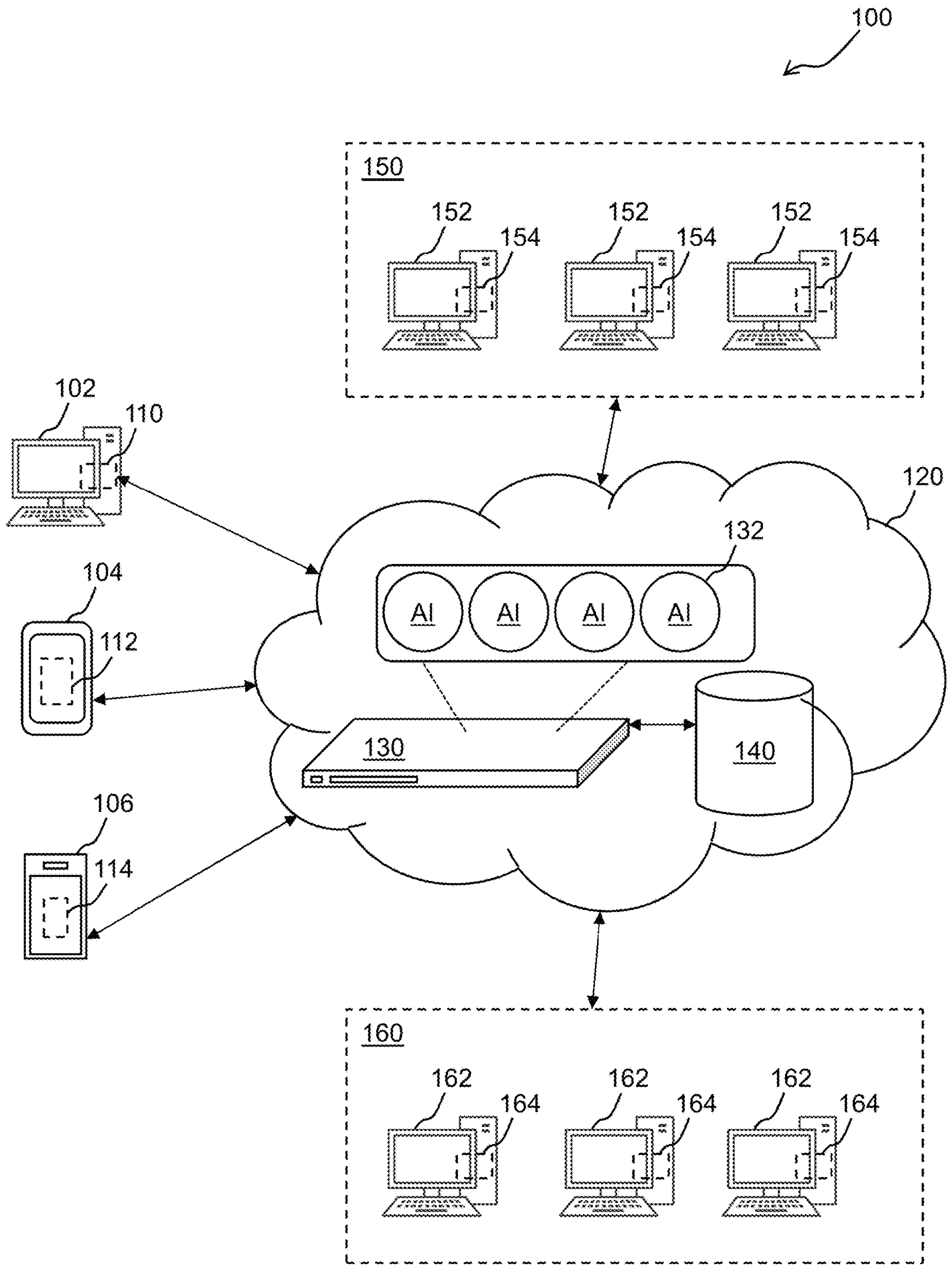

100
150
152
154
102
110
120
132
AI
104
112
130
140
106
114
160
162
164

FIG. 1

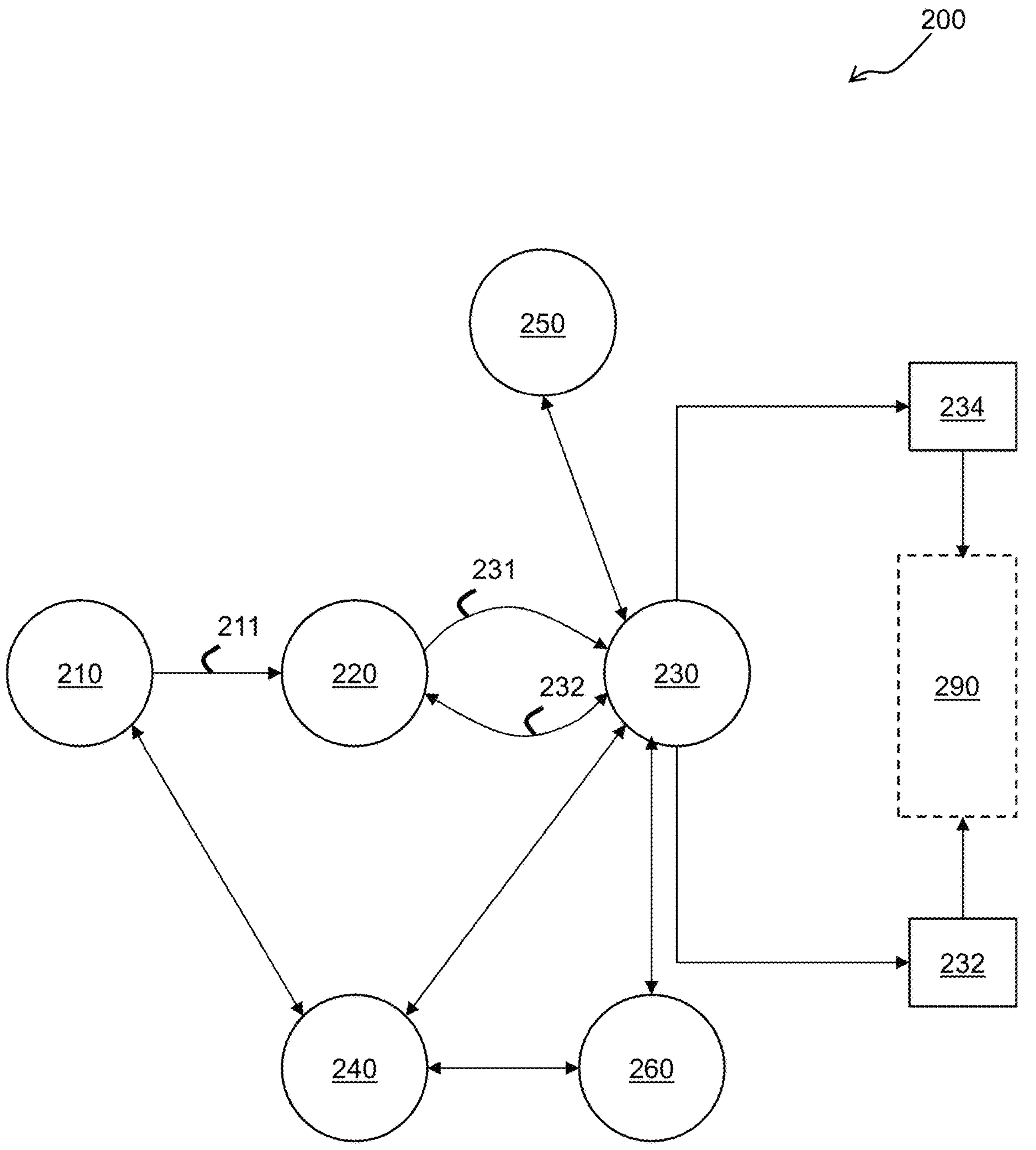


FIG. 2

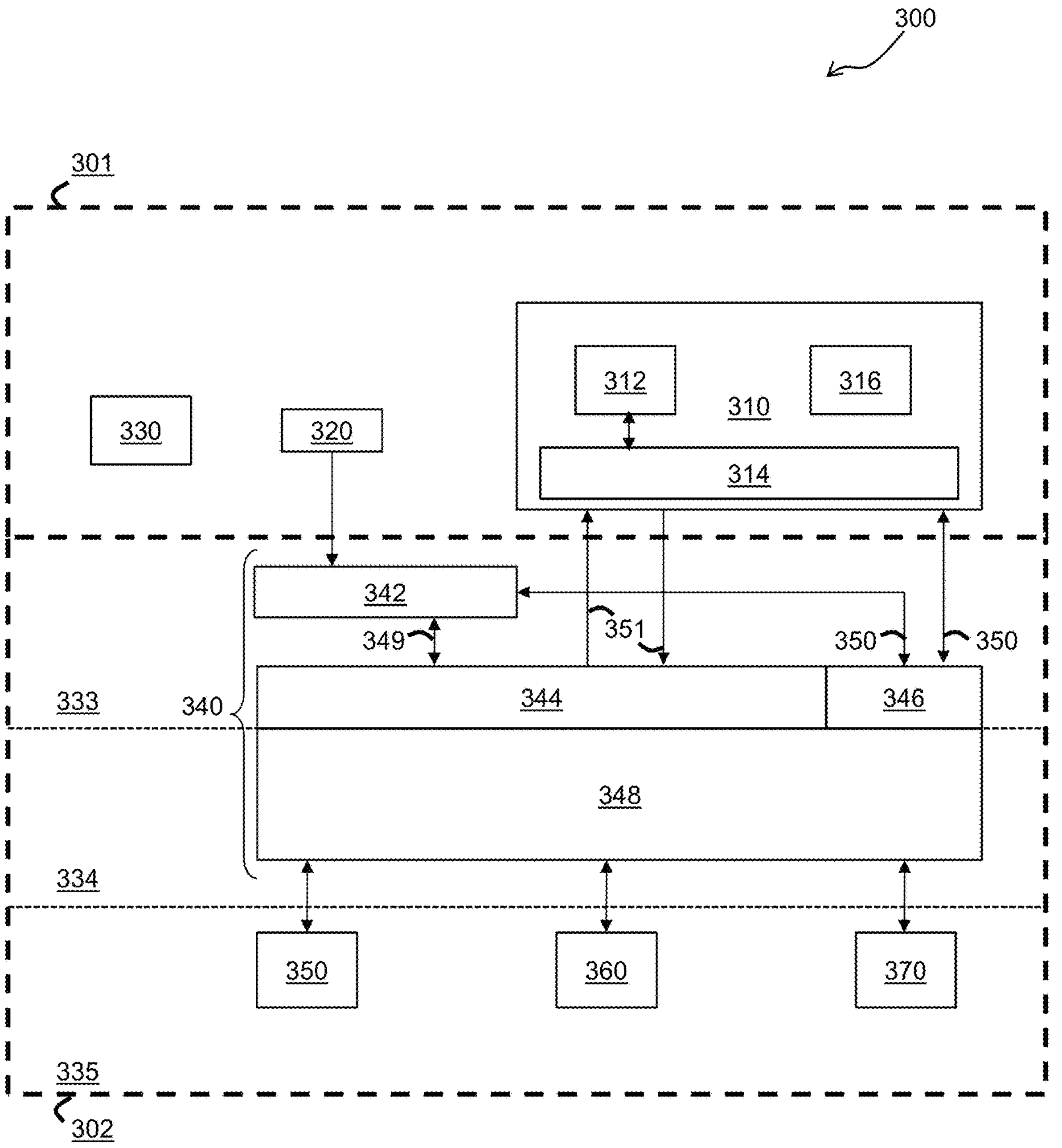

300
301
330
320
312
316
310
314
342
349
351
350
350
333
340
344
346
348
334
350
360
370
335
302

FIG. 3

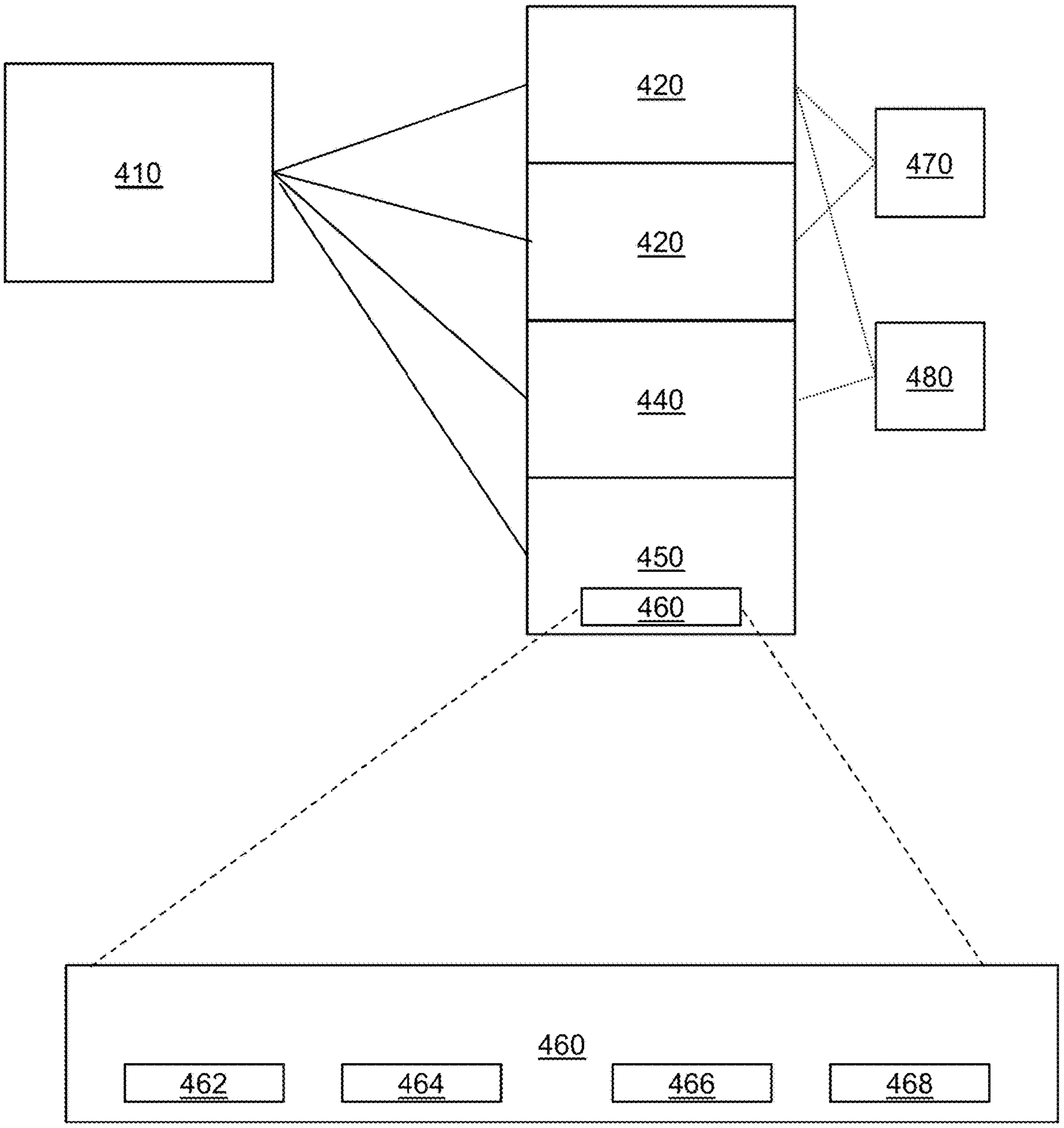


FIG. 4

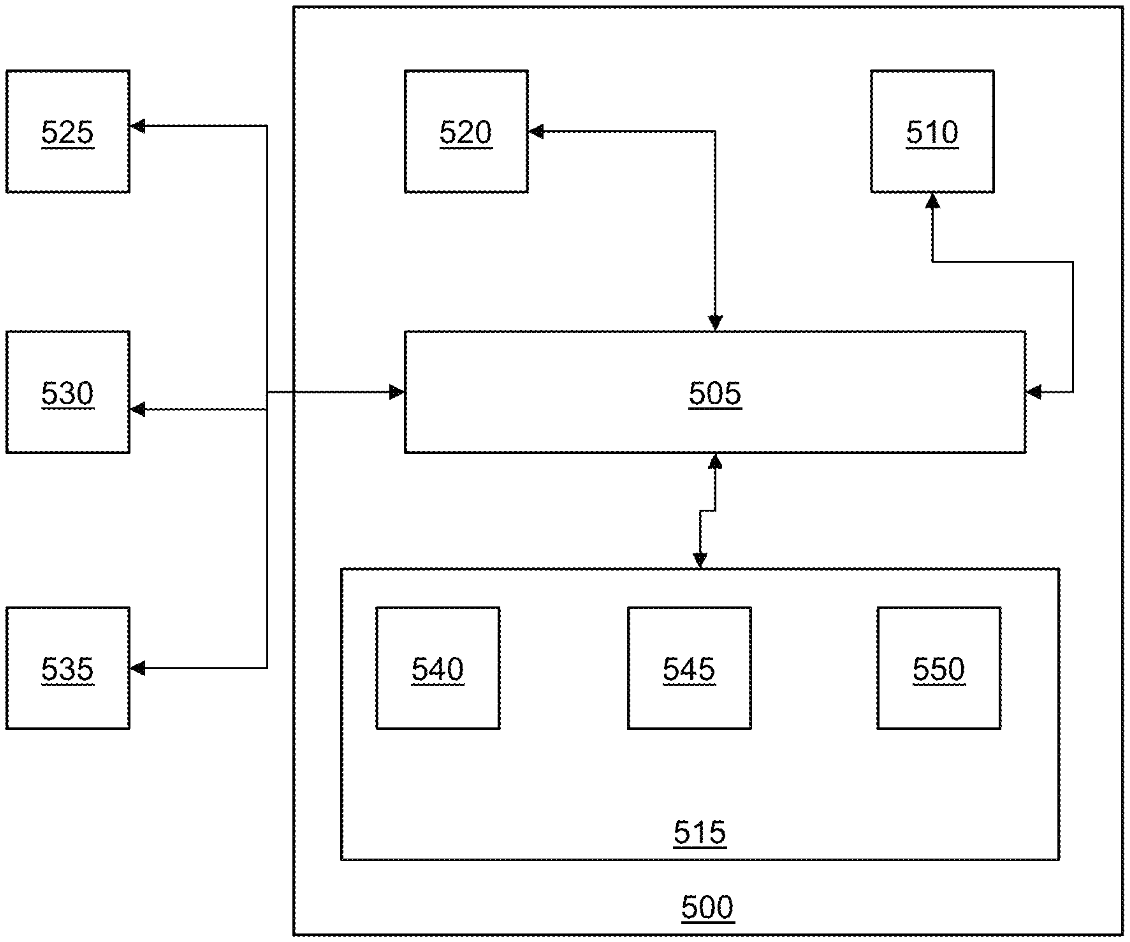


FIG. 5

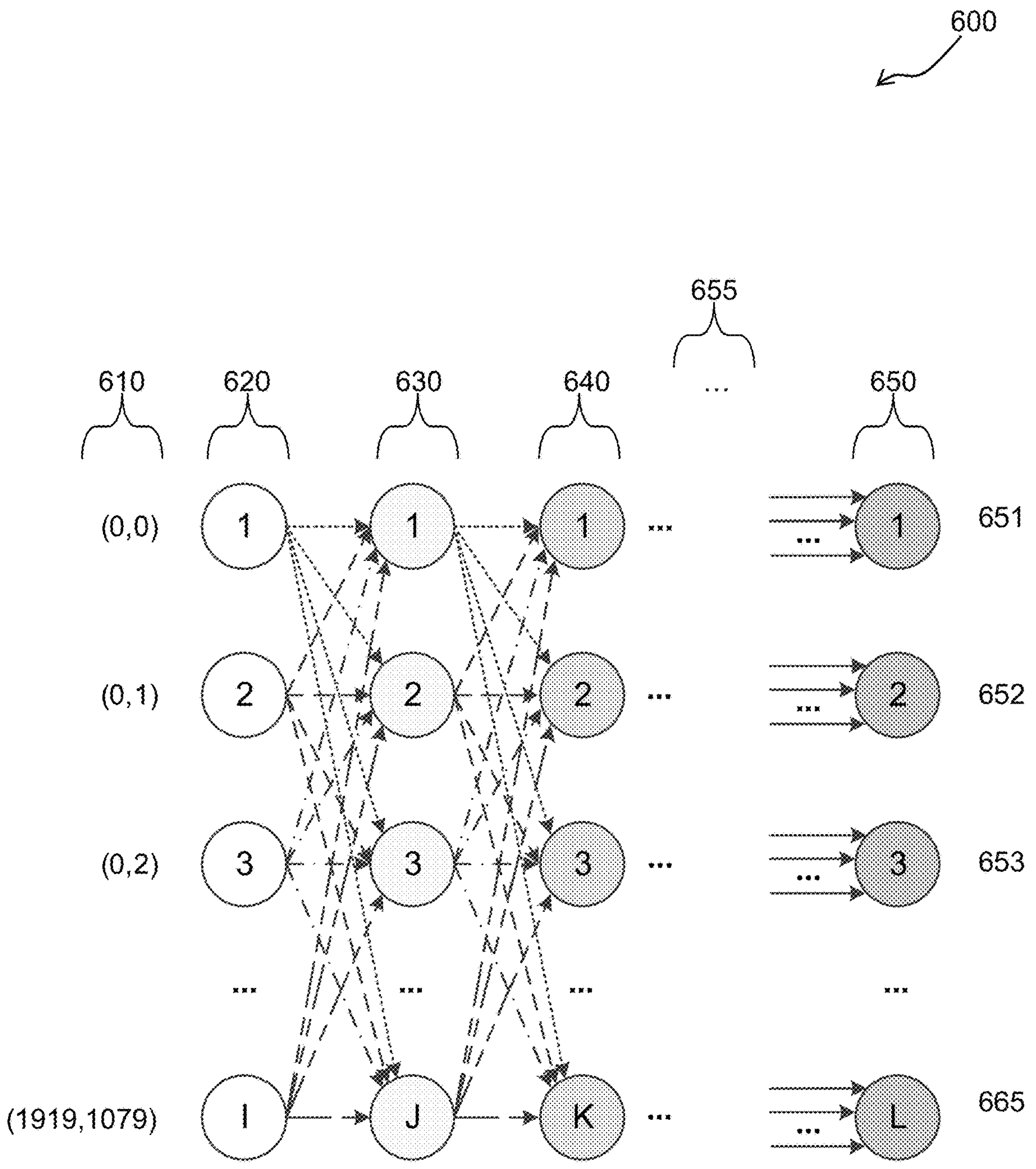

600
655
610
620
630
640
650
(0,0)
(0,1)
(0,2)
(1919,1079)
1
2
3
I
J
K
L
651
652
653
665

FIG. 6

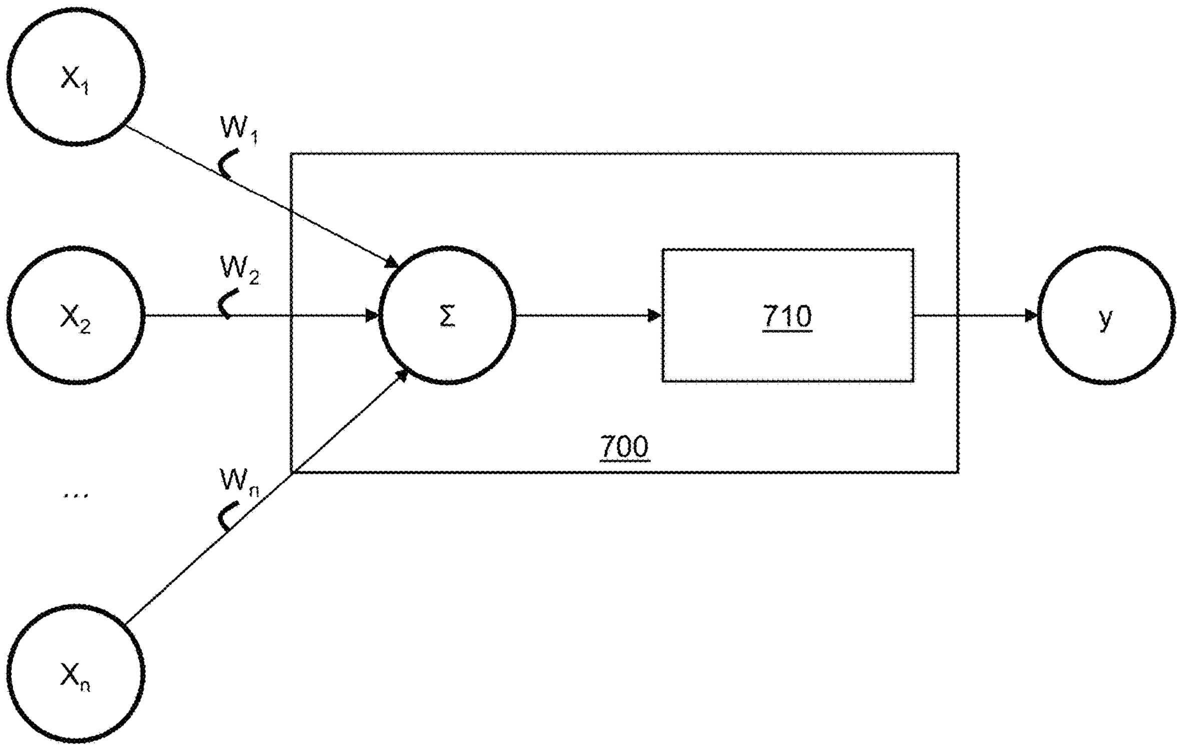


FIG. 7

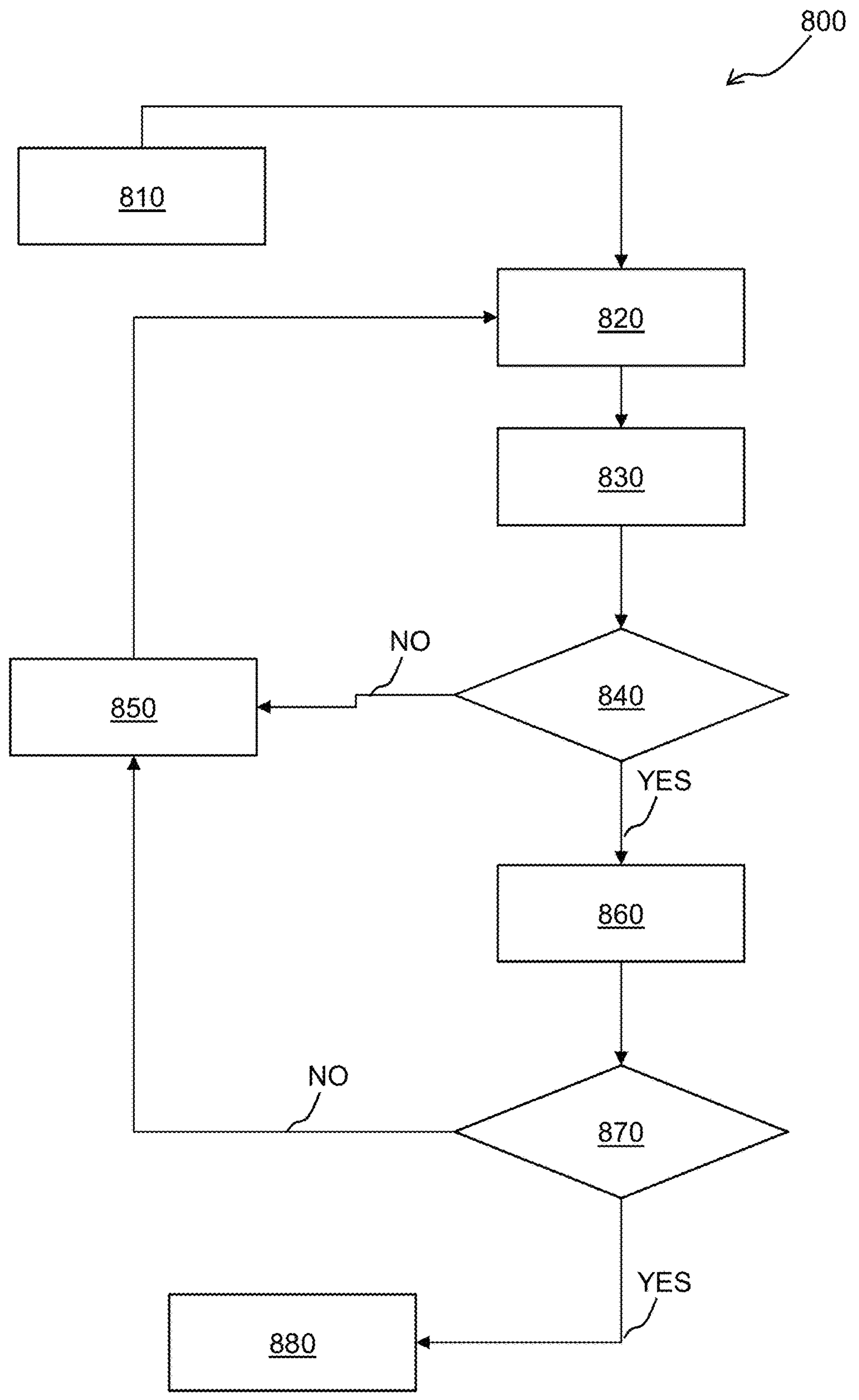


FIG. 8

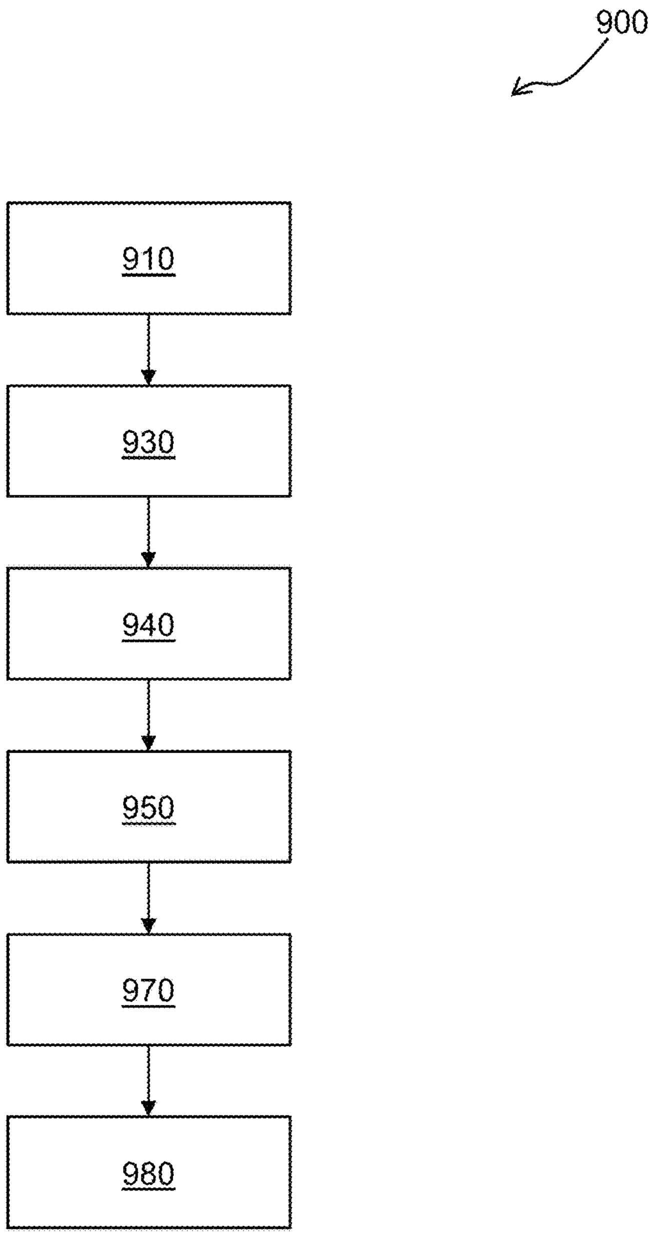


FIG. 9

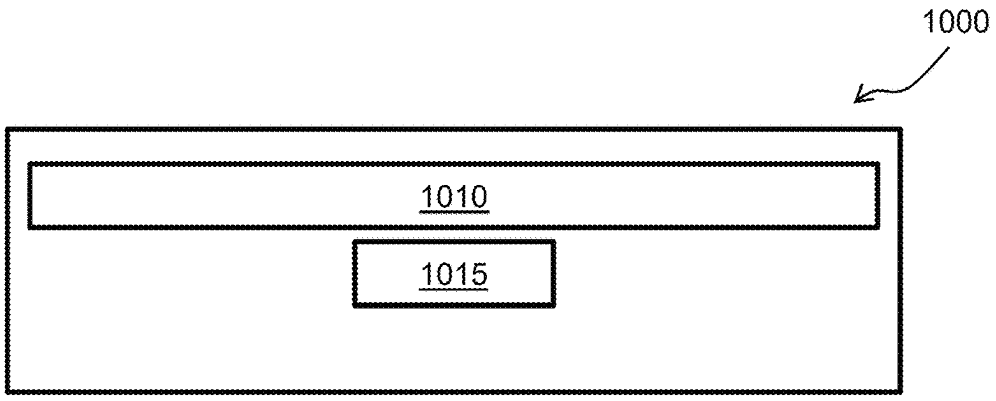

1000
1010
1015

FIG. 10

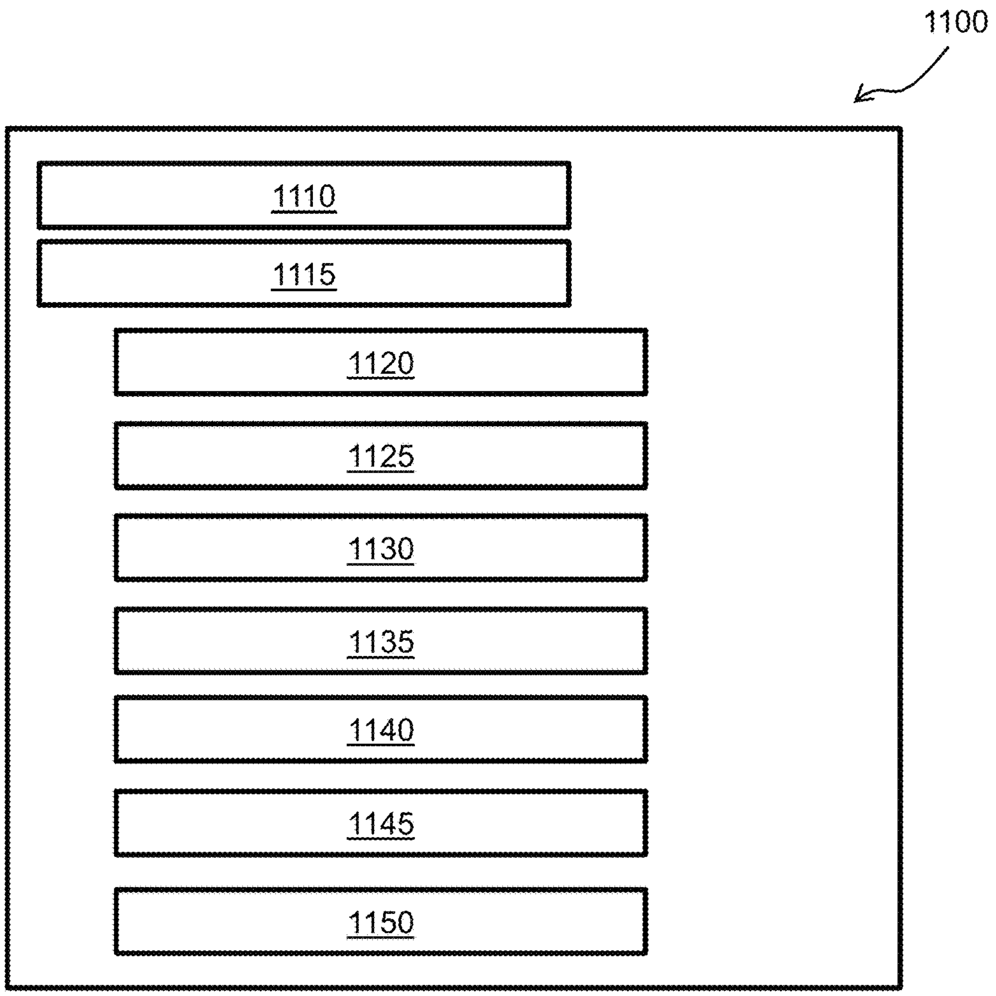

1100
1110
1115
1120
1125
1130
1135
1140
1145
1150

FIG. 11

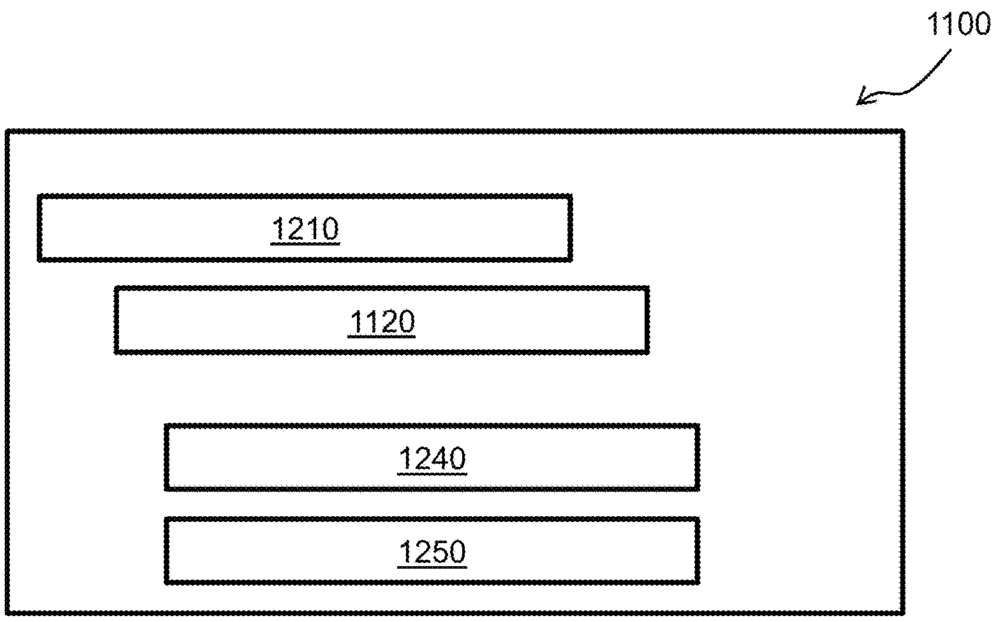

1100
1210
1120
1240
1250

FIG. 12

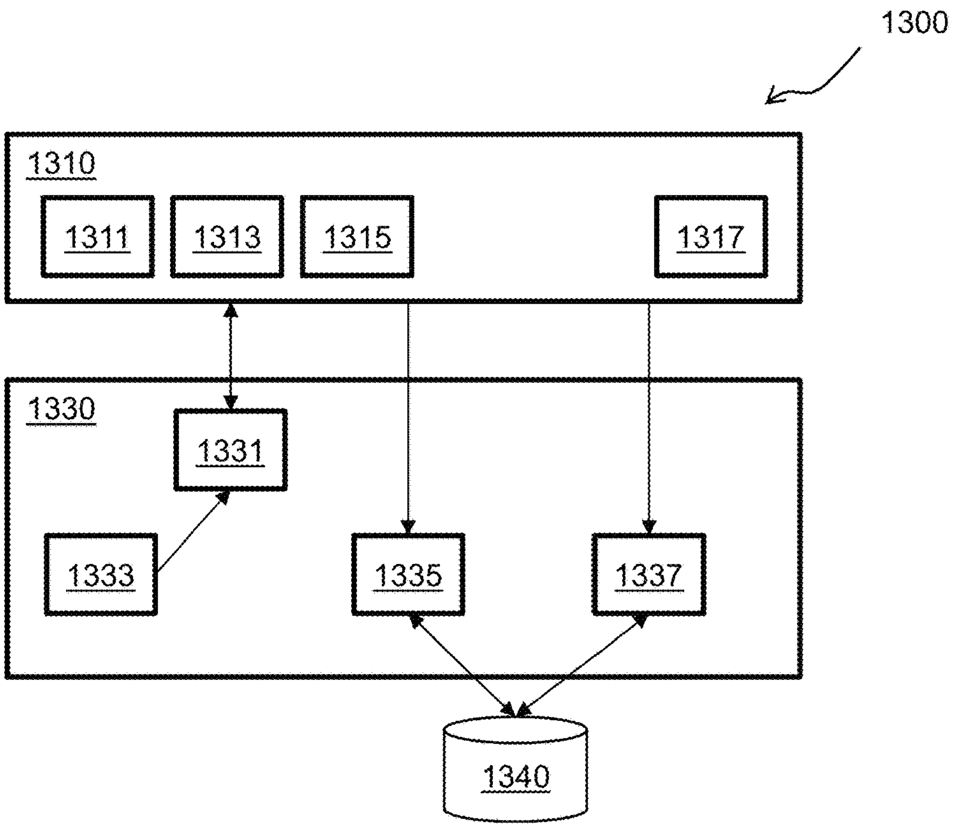

1300
1310
1311
1313
1315
1317
1330
1331
1333
1335
1337
1340

FIG. 13

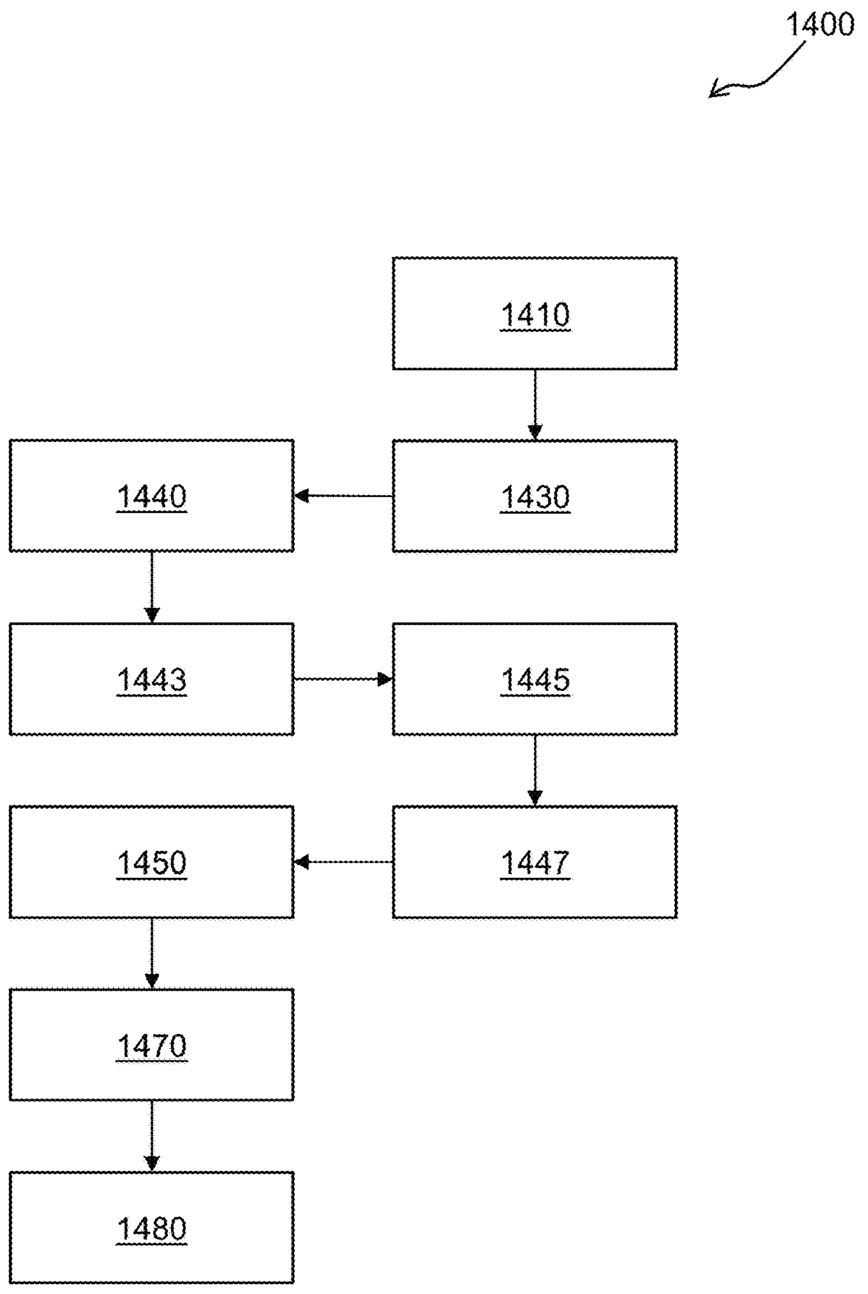


FIG. 14

TYPE CACHE FOR PACKAGE MANAGEMENT OF ROBOTIC PROCESS AUTOMATIONS FIELD

FIELD

This disclosure generally relates to automation, and more specifically, to type cache for package management of robotic process automation (RPA).

BACKGROUND

RPA is a category of software that requires activities. Activities, said another way, are the building blocks of RPA. In this regard, Activities construct a workflow in a designer application to perform a RPA. Activities are 'shipped', i.e., provided to a designer, using a package manager and framework (e.g., NuGet packages and a NuGet framework), which are delivered into the designer application.

To leverage anyone of these activities in an automation, the designer must install the package manager and framework (i.e., using NuGet Restoration, which is specific to the NuGet framework) within the designer application. Note that NuGet Restoration is slow operation by nature because the NuGet Restoration has to query different services multiple times to construct a dependency graph. Further, installation provides a bottleneck for the designing RPAs as the designer will not be able to know activities that are available from the package manager and framework until the installation is finished.

For example, the NuGet Restoration manages all the dependencies and sub-dependencies of the NuGet packages, as well as resolves the dependencies on behalf of user. Unfortunately, installing packages, such as by NuGet Restoration, resolves the dependencies on behalf of the designer and leaves a problem for the designer in that the designer will not be able to know the activities are available from the NuGet packages and a NuGet framework until the NuGet Restoration is finished.

SUMMARY

According to one or more embodiments, a method executed by a type cache service implemented as a computer program within a computing environment is provided. The method includes scanning one or more feeds publishing one or more unpacked packages; automatically detecting at least new activities on the one or more feeds; indexing the at least new activities and all previously detected activities according to type; and generating a type cache for the one or more unpacked packages according to the indexing of the at least new activities and all previously detected activities.

According to one or more embodiments, a method executed by a type cache service implemented as a computer program within a computing environment is provided. The method includes parsing indexed entries for one or more unpacked packages of a type cache; determining a best common denominator of the one or more unpacked packages based on the type cache to utilize in a dependency graph; and resolving, by the type cache service, dependencies of the dependency graph based on the best common denominator.

BRIEF DESCRIPTION OF THE DRAWINGS

In order that the advantages of certain embodiments herein will be readily understood, a more particular descrip-

tion will be rendered by reference to specific embodiments that are illustrated in the appended drawings. While it should be understood that these drawings depict only typical embodiments and are not therefore to be considered to be limiting of its scope, the one or more embodiments herein will be described and explained with additional specificity and detail through the use of the accompanying drawings, in which:

FIG. **1** depicts an architectural diagram illustrating an automation system according to one or more embodiments.

FIG. **2** depicts an architectural diagram illustrating a RPA system according to one or more embodiments.

FIG. **3** depicts an architectural diagram illustrating a deployed RPA system, according to one or more embodiments.

FIG. **4** depicts an architectural diagram illustrating relationships between a designer, activities, and drivers according to one or more embodiments.

FIG. **5** depicts an architectural diagram illustrating a computing system according to one or more embodiments.

FIG. **6** illustrates an example of a neural network that has been trained to recognize graphical elements in an image according to one or more embodiments.

FIG. **7** illustrates an example of a neuron according to one or more embodiments.

FIG. **8** depicts a flowchart illustrating a process for training AI/ML model(s) according to one or more embodiments.

FIG. **9** depicts a flowchart illustrating a process for training AI/ML model(s) according to one or more embodiments.

FIG. **10** depicts an interface according to one or more embodiments.

FIG. **11** depicts an interface according to one or more embodiments.

FIG. **12** depicts an interface according to one or more embodiments.

FIG. **13** depicts an architectural diagram illustrating a computing environment according to one or more embodiments according to one or more embodiments.

FIG. **14** depicts a flowchart illustrating a process according to one or more embodiments.

Unless otherwise indicated, similar reference characters denote corresponding features consistently throughout the attached drawings.

DETAILED DESCRIPTION OF THE EMBODIMENTS

Generally, embodiments herein pertain to automation. More specifically, embodiments herein pertain to type cache for package management of automations/RPAs. Implementing the type cache for package management can be performed by a computing system and/or a type cache service (as described herein).

As noted, installing packages, such as by NuGet Restoration, leaves a problem for the designer in that the designer will not be able to know the activities that are available from the packages and the framework until the installation is finished. For example, in current RPA studio software, a designer needs to search for a package to use an activity in a workflow. In turn, the designer generally is required to have the knowledge of which package consists of what activities before searching for that activity. Once the package is found, the designer installs that entire package and, then, searches for the activity required by the workflow.

According to one or more embodiments, the computing system and/or a type cache service proactively scans all folders and feeds (e.g., a feed is a mechanism to receive data and updated data from data sources, such as NuGet feeds) that are subject to scanning and automatically detects new activities/items. Type cache can include an auxiliary collection of types of activities/items/.net objects/assemblies/etc., as well as intelligence data and keywords corresponding thereto, within one or more packages. The type cache service works in background to generate a type cache from the scanned and detected new activities/items to provide these new activities/items in a searchable form before installation (e.g., before NuGet Restoration). One or more technical effects, advantages, or benefits of implementing the type cache for package management of automations/RPAs include the computing system providing search capabilities for known types (e.g., case activities) that contain a given keyword (e.g., localized) in all packages (whether unpacked or installed) that are published in a list of feeds.

Further, the computing system retrieves a list of required dynamic-link libraries (DLLs) from a dependency graph (e.g., a NuGet dependency graph) required to use that type, which greatly improves load times of necessary DLLs to use a desired type by skipping package installation processes (e.g., the NuGet Restoration, which is relatively slow in comparison). Additionally, while a local storage can be created (to cache known packages) by package installation processes by design, in some situations (e.g., web, lambda, etc.) local cache storage is not possible. In turn, the computing system can solve this technical issue by leveraging the type cache to act as a local cache. Thus, the computing system provides a type cache service to make activities within a package discoverable before restoration.

FIG. **1** is an architectural diagram illustrating a hyper-automation system **100**, according to one or more embodiments. "Hyper-automation," as used herein, refers to automation systems that bring together components of process automation, integration tools, and technologies that amplify the ability to automate work. For instance, RPA may be used at the core of a hyper-automation system in some embodiments, and in certain embodiments, automation capabilities may be expanded with artificial intelligence and/or machine (AI/ML), process mining, analytics, and/or other advanced tools. As the hyper-automation system learns processes, trains AI/ML models, and employs analytics, for example, more and more knowledge work may be automated, and computing systems in an organization, e.g., both those used by individuals and those that run autonomously, may all be engaged to be participants in the hyper-automation process. Hyper-automation systems of some embodiments allow users and organizations to efficiently and effectively discover, understand, and scale automations.

Hyper-automation system **100** includes user computing systems, such as desktop computer **102**, tablet **104**, and smart phone **106**. However, any desired computing system may be used without deviating from the scope of one or more embodiments herein including, but not limited to, smart watches, laptop computers, servers, Internet-of-Things (IoT) devices, etc. Also, while three user computing systems are shown in FIG. **1**, any suitable number of computing systems may be used without deviating from the scope of the one or more embodiments herein. For instance, in some embodiments, dozens, hundreds, thousands, or millions of computing systems may be used. The user computing systems may be actively used by a user or run automatically without much or any user input.

Each computing system **102**, **104**, **106** has respective automation process(es) **110**, **112**, **114** running thereon. Automation process(es) **102**, **104**, **106** may include, but are not limited to, RPA robots, part of an operating system, downloadable application(s) for the respective computing system, any other suitable software and/or hardware, or any combination of these without deviating from the scope of the one or more embodiments herein. In some embodiments, one or more of process(es) **110**, **112**, **114** may be listeners. Listeners may be RPA robots, part of an operating system, a downloadable application for the respective computing system, or any other software and/or hardware without deviating from the scope of the one or more embodiments herein. Indeed, in some embodiments, the logic of the listener(s) is implemented partially or completely via physical hardware.

Listeners monitor and record data pertaining to user interactions with respective computing systems and/or operations of unattended computing systems and send the data to a core hyper-automation system **120** via a network (e.g., a local area network (LAN), a mobile communications network, a satellite communications network, the Internet, any combination thereof, etc.). The data may include, but is not limited to, which buttons were clicked, where a mouse was moved, the text that was entered in a field, that one window was minimized and another was opened, the application associated with a window, etc. In certain embodiments, the data from the listeners may be sent periodically as part of a heartbeat message. In some embodiments, the data may be sent to core hyper-automation system **120** once a predetermined amount of data has been collected, after a predetermined time period has elapsed, or both. One or more servers, such as server **130**, receive and store data from the listeners in a database, such as database **140**.

Automation processes may execute the logic developed in workflows during design time. In the case of RPA, workflows may include a set of steps, defined herein as "activities," that are executed in a sequence or some other logical flow. Each activity may include an action, such as clicking a button, reading a file, writing to a log panel, etc. In some embodiments, workflows may be nested or embedded.

Long-running workflows for RPA in some embodiments are master projects that support service orchestration, human intervention, and long-running transactions in unattended environments. See U.S. Pat. No. 10,860,905, which is incorporated by reference for all it contains. Human intervention comes into play when certain processes require human inputs to handle exceptions, approvals, or validation before proceeding to the next step in the activity. In this situation, the process execution is suspended, freeing up the RPA robots until the human task completes.

A long-running workflow may support workflow fragmentation via persistence activities and may be combined with invoke process and non-user interaction activities, orchestrating human tasks with RPA robot tasks. In some embodiments, multiple or many computing systems may participate in executing the logic of a long-running workflow. The long-running workflow may run in a session to facilitate speedy execution. In some embodiments, long-running workflows may orchestrate background processes that may contain activities performing Application Programming Interface (API) calls and running in the long-running workflow session. These activities may be invoked by an invoke process activity in some embodiments. A process with user interaction activities that runs in a user session may be called by starting a job from a conductor activity (conductor described in more detail later herein). The user may interact through tasks that require forms to be completed in the conductor in some embodiments. Activities may be included that cause the RPA robot to wait for a form task to be completed and then resume the long-running workflow.

One or more of automation process(es) **110**, **112**, **114** is in communication with core hyper-automation system **120**. In some embodiments, core hyper-automation system **120** may run a conductor application on one or more servers, such as server **130**. While one server **130** is shown for illustration purposes, multiple or many servers that are proximate to one another or in a distributed architecture may be employed without deviating from the scope of the one or more embodiments herein. For instance, one or more servers may be provided for conductor functionality, AI/ML model serving, authentication, governance, and/or any other suitable functionality without deviating from the scope of the one or more embodiments herein. In some embodiments, core hyper-automation system **120** may incorporate or be part of a public cloud architecture, a private cloud architecture, a hybrid cloud architecture, etc. In certain embodiments, core hyper-automation system **120** may host multiple software-based servers on one or more computing systems, such as server **130**. In some embodiments, one or more servers of core hyper-automation system **120**, such as server **130**, may be implemented via one or more virtual machines (VMs).

In some embodiments, one or more of automation process(es) **110**, **112**, **114** may call one or more AI/ML models **132** deployed on or accessible by core hyper-automation system **120**. AI/ML models **132** may be trained for any suitable purpose without deviating from the scope of the one or more embodiments herein, as will be discussed in more detail herein. Two or more of AI/ML models **132** may be chained in some embodiments (e.g., in series, in parallel, or a combination thereof) such that they collectively provide collaborative output(s). AI/ML models **132** may perform or assist with computer vision (CV), optical character recognition (OCR), document processing and/or understanding, semantic learning and/or analysis, analytical predictions, process discovery, task mining, testing, automatic RPA workflow generation, sequence extraction, clustering detection, audio-to-text translation, any combination thereof, etc. However, any desired number and/or type(s) of AI/ML models may be used without deviating from the scope of the one or more embodiments herein. Using multiple AI/ML models may allow the system to develop a global picture of what is happening on a given computing system, for example. For instance, one AI/ML model could perform OCR, another could detect buttons, another could compare sequences, etc. Patterns may be determined individually by an AI/ML model or collectively by multiple AI/ML models. In certain embodiments, one or more AI/ML models are deployed locally on at least one of computing systems **102**, **104**, **106**.

In some embodiments, multiple AI/ML models **132** may be used. Each AI/ML model **132** is an algorithm (or model) that runs on the data, and the AI/ML model itself may be a deep learning neural network (DLNN) of trained artificial "neurons" that are trained in training data, for example. In some embodiments, AI/ML models **132** may have multiple layers that perform various functions, such as statistical modeling (e.g., hidden Markov models (HMMs)), and utilize deep learning techniques (e.g., long short term memory (LSTM) deep learning, encoding of previous hidden states, etc.) to perform the desired functionality.

Hyper-automation system **100** may provide four main groups of functionality in some embodiments: (1) discovery; (2) building automations; (3) management; and (4) engagement. Automations (e.g., run on a user computing system, a server, etc.) may be run by software robots, such as RPA robots, in some embodiments. For instance, attended robots, unattended robots, and/or test robots may be used. Attended robots work with users to assist them with tasks (e.g., via UiPath Assistant™). Unattended robots work independently of users and may run in the background, potentially without user knowledge. Test robots are unattended robots that run test cases against applications or RPA workflows. Test robots may be run on multiple computing systems in parallel in some embodiments.

The discovery functionality may discover and provide automatic recommendations for different opportunities of automations of business processes. Such functionality may be implemented by one or more servers, such as server **130**. The discovery functionality may include providing an automation hub, process mining, task mining, and/or task capture in some embodiments. The automation hub (e.g., UiPath Automation Hub™) may provide a mechanism for managing automation rollout with visibility and control. Automation ideas may be crowdsourced from employees via a submission form, for example. Feasibility and return on investment (ROI) calculations for automating these ideas may be provided, documentation for future automations may be collected, and collaboration may be provided to get from automation discovery to build-out faster.

Process mining (e.g., via UiPath Automation Cloud™ and/or UiPath AI Center™) refers to the process of gathering and analyzing the data from applications (e.g., enterprise resource planning (ERP) applications, customer relation management (CRM) applications, email applications, call center applications, etc.) to identify what end-to-end processes exist in an organization and how to automate them effectively, as well as indicate what the impact of the automation will be. This data may be gleaned from user computing systems **102**, **104**, **106** by listeners, for example, and processed by servers, such as server **130**. One or more AI/ML models **132** may be employed for this purpose in some embodiments. This information may be exported to the automation hub to speed up implementation and avoid manual information transfer. The goal of process mining may be to increase business value by automating processes within an organization. Some examples of process mining goals include, but are not limited to, increasing profit, improving customer satisfaction, regulatory and/or contractual compliance, improving employee efficiency, etc.

Task mining (e.g., via UiPath Automation Cloud™ and/or UiPath AI Center™) identifies and aggregates workflows (e.g., employee workflows), and then applies AI to expose patterns and variations in day-to-day tasks, scoring such tasks for ease of automation and potential savings (e.g., time and/or cost savings). One or more AI/ML models **132** may be employed to uncover recurring task patterns in the data. Repetitive tasks that are ripe for automation may then be identified. This information may initially be provided by listeners and analyzed on servers of core hyper-automation system **120**, such as server **130**, in some embodiments. The findings from task mining (e.g., extensive application markup language (XAML) process data) may be exported to process documents or to a designer application such as UiPath Studio™ to create and deploy automations more rapidly.

Task mining in some embodiments may include taking screenshots with user actions (e.g., mouse click locations, keyboard inputs, application windows and graphical elements the user was interacting with, timestamps for the interactions, etc.), collecting statistical data (e.g., execution time, number of actions, text entries, etc.), editing and annotating screenshots, specifying types of actions to be recorded, etc.

Task capture (e.g., via UiPath Automation Cloud™ and/or UiPath AI Center™) automatically documents attended processes as users work or provides a framework for unattended processes. Such documentation may include desired tasks to automate in the form of process definition documents (PDDs), skeletal workflows, capturing actions for each part of a process, recording user actions and automatically generating a comprehensive workflow diagram including the details about each step, Microsoft Word® documents, XAML files, and the like. Build-ready workflows may be exported directly to a designer application in some embodiments, such as UiPath Studio™. Task capture may simplify the requirements gathering process for both subject matter experts explaining a process and Center of Excellence (CoE) members providing production-grade automations.

Building automations may be accomplished via a designer application (e.g., UiPath Studio™, UiPath StudioX™, or UiPath Web™). For instance, RPA developers of an PA development facility **150** may use RPA designer applications **154** of computing systems **152** to build and test automations for various applications and environments, such as web, mobile, SAP®, and virtualized desktops. API integration may be provided for various applications, technologies, and platforms. Predefined activities, drag-and-drop modeling, and a workflow recorder, may make automation easier with minimal coding. Document understanding functionality may be provided via drag-and-drop AI skills for data extraction and interpretation that call one or more AI/ML models **132**. Such automations may process virtually any document type and format, including tables, checkboxes, signatures, and handwriting. When data is validated or exceptions are handled, this information may be used to retrain the respective AI/ML models, improving their accuracy over time.

An integration service may allow developers to seamlessly combine user interface (UI) automation with API automation, for example. Automations may be built that require APIs or traverse both API and non-API applications and systems. A repository (e.g., UiPath Object Repository™) or marketplace (e.g., UiPath Marketplace™) for pre-built RPA and AI templates and solutions may be provided to allow developers to automate a wide variety of processes more quickly. Thus, when building automations, hyper-automation system **100** may provide user interfaces, development environments, API integration, pre-built and/or custom-built AI/ML models, development templates, integrated development environments (IDEs), and advanced AI capabilities. Hyper-automation system **100** enables development, deployment, management, configuration, monitoring, debugging, and maintenance of RPA robots in some embodiments, which may provide automations for hyper-automation system **100**.

In some embodiments, components of hyper-automation system **100**, such as designer application(s) and/or an external rules engine, provide support for managing and enforcing governance policies for controlling various functionality provided by hyper-automation system **100**. Governance is the ability for organizations to put policies in place to prevent users from developing automations (e.g., RPA robots) capable of taking actions that may harm the organization, such as violating the E.U. General Data Protection Regulation (GDPR), the U.S. Health Insurance Portability and Accountability Act (HIPAA), third party application terms of service, etc. Since developers may otherwise create automations that violate privacy laws, terms of service, etc. while performing their automations, some embodiments implement access control and governance restrictions at the robot and/or robot design application level. This may provide an added level of security and compliance into the automation process development pipeline in some embodiments by preventing developers from taking dependencies on unapproved software libraries that may either introduce security risks or work in a way that violates policies, regulations, privacy laws, and/or privacy policies. See U.S. Nonprovisional patent application Ser. No. 16/924,499, which is incorporated by reference for all it contains.

The management functionality may provide management, deployment, and optimization of automations across an organization. The management functionality may include orchestration, test management, AI functionality, and/or insights in some embodiments. Management functionality of hyper-automation system **100** may also act as an integration point with third-party solutions and applications for automation applications and/or RPA robots. The management capabilities of hyper-automation system **100** may include, but are not limited to, facilitating provisioning, deployment, configuration, queuing, monitoring, logging, and interconnectivity of RPA robots, among other things.

A conductor application, such as UiPath Orchestrator™ (which may be provided as part of the UiPath Automation Cloud™ in some embodiments, or on premises, in VMs, in a private or public cloud, in a Linux™ VM, or as a cloud native single container suite via UiPath Automation Suite™), provides orchestration capabilities to deploy, monitor, optimize, scale, and ensure security of RPA robot deployments. A test suite (e.g., UiPath Test Suite™) may provide test management to monitor the quality of deployed automations. The test suite may facilitate test planning and execution, meeting of requirements, and defect traceability. The test suite may include comprehensive test reporting.

Analytics software (e.g., UiPath Insights™) may track, measure, and manage the performance of deployed automations. The analytics software may align automation operations with specific key performance indicators (KPIs) and strategic outcomes for an organization. The analytics software may present results in a dashboard format for better understanding by human users.

A data service (e.g., UiPath Data Service™) may be stored in database **140**, for example, and bring data into a single, scalable, secure place with a drag-and-drop storage interface. Some embodiments may provide low-code or no-code data modeling and storage to automations while ensuring seamless access, enterprise-grade security, and scalability of the data. AI functionality may be provided by an AI center (e.g., UiPath AI Center™), which facilitates incorporation of AI/ML models into automations. Pre-built AI/ML models, model templates, and various deployment options may make such functionality accessible even to those who are not data scientists. Deployed automations (e.g., RPA robots) may call AI/ML models from the AI center, such as AI/ML models **132**. Performance of the AI/ML models may be monitored, and be trained and improved using human-validated data, such as that provided by data review center **160**. Human reviewers may provide labeled data to core hyper-automation system **120** via a review application **152** on computing systems **154**. For instance, human reviewers may validate that predictions by AI/ML models **132** are accurate or provide corrections otherwise. This dynamic input may then be saved as training data for retraining AI/ML models **132**, and may be stored in a database such as database **140**, for example. The AI center may then schedule and execute training jobs to train the new versions of the AI/ML models using the training data. Both positive and negative examples may be stored and used for retraining of AI/ML models **132**.

The engagement functionality engages humans and automations as one team for seamless collaboration on desired processes. Low-code applications may be built (e.g., via UiPath Apps™) to connect browser tabs and legacy software, even that lacking APIs in some embodiments. Applications may be created quickly using a web browser through a rich library of drag-and-drop controls, for instance. An application can be connected to a single automation or multiple automations.

An action center (e.g., UiPath Action Center™) provides a straightforward and efficient mechanism to hand off processes from automations to humans, and vice versa. Humans may provide approvals or escalations, make exceptions, etc. The automation may then perform the automatic functionality of a given workflow.

A local assistant may be provided as a launchpad for users to launch automations (e.g., UiPath Assistant™). This functionality may be provided in a tray provided by an operating system, for example, and may allow users to interact with RPA robots and RPA robot-powered applications on their computing systems. An interface may list automations approved for a given user and allow the user to run them. These may include ready-to-go automations from an automation marketplace, an internal automation store in an automation hub, etc. When automations run, they may run as a local instance in parallel with other processes on the computing system so users can use the computing system while the automation performs its actions. In certain embodiments, the assistant is integrated with the task capture functionality such that users can document their soon-to-be-automated processes from the assistant launchpad.

Chatbots (e.g., UiPath Chatbots™), social messaging applications, and/or voice commands may enable users to run automations. This may simplify access to information, tools, and resources users need in order to interact with customers or perform other activities. Conversations between people may be readily automated, as with other processes. Trigger RPA robots kicked off in this manner may perform operations such as checking an order status, posting data in a CRM, etc., potentially using plain language commands.

End-to-end measurement and government of an automation program at any scale may be provided by hyper-automation system **100** in some embodiments. Per the above, analytics may be employed to understand the performance of automations (e.g., via UiPath Insights™). Data modeling and analytics using any combination of available business metrics and operational insights may be used for various automated processes. Custom-designed and pre-built dashboards allow data to be visualized across desired metrics, new analytical insights to be discovered, performance indicators to be tracked, ROI to be discovered for automations, telemetry monitoring to be performed on user computing systems, errors and anomalies to be detected, and automations to be debugged. An automation management console (e.g., UiPath Automation Ops™) may be provided to manage automations throughout the automation lifecycle. An organization may govern how automations are built, what users can do with them, and which automations users can access.

Hyper-automation system **100** provides an iterative platform in some embodiments. Processes can be discovered, automations can be built, tested, and deployed, performance may be measured, use of the automations may readily be provided to users, feedback may be obtained, AI/ML models may be trained and retrained, and the process may repeat itself. This facilitates a more robust and effective suite of automations.

FIG. **2** is an architectural diagram illustrating an RPA system **200**, according to one or more embodiments. In some embodiments, RPA system **200** is part of hyper-automation system **100** of FIG. **1**. RPA system **200** includes a designer **210** that allows a developer to design and implement workflows. Designer **210** may provide a solution for application integration, as well as automating third-party applications, administrative Information Technology (IT) tasks, and business IT processes. Designer **210** may facilitate development of an automation project, which is a graphical representation of a business process. Simply put, designer **210** facilitates the development and deployment (as represented by arrow **211**) of workflows and robots. In some embodiments, designer **210** may be an application that runs on a user's desktop, an application that runs remotely in a VM, a web application, etc.

The automation project enables automation of rule-based processes by giving the developer control of the execution order and the relationship between a custom set of steps developed in a workflow, defined herein as "activities" per the above. One commercial example of an embodiment of designer **210** is UiPath Studio™. Each activity may include an action, such as clicking a button, reading a file, writing to a log panel, etc. In some embodiments, workflows may be nested or embedded.

Some types of workflows may include, but are not limited to, sequences, flowcharts, Finite State Machines (FSMs), and/or global exception handlers. Sequences may be particularly suitable for linear processes, enabling flow from one activity to another without cluttering a workflow. Flowcharts may be particularly suitable to more complex business logic, enabling integration of decisions and connection of activities in a more diverse manner through multiple branching logic operators. FSMs may be particularly suitable for large workflows. FSMs may use a finite number of states in their execution, which are triggered by a condition (i.e., transition) or an activity. Global exception handlers may be particularly suitable for determining workflow behavior when encountering an execution error and for debugging processes.

Once a workflow is developed in designer **210**, execution of business processes is orchestrated by conductor **220**, which orchestrates one or more robots **230** that execute the workflows developed in designer **210**. One commercial example of an embodiment of conductor **220** is UiPath Orchestrator™. Conductor **220** facilitates management of the creation, monitoring, and deployment of resources in an environment. Conductor **220** may act as an integration point with third-party solutions and applications. Per the above, in some embodiments, conductor **220** may be part of core hyper-automation system **120** of FIG. **1**.

Conductor **220** may manage a fleet of robots **230**, connecting and executing (as represented by arrow **231**) robots **230** from a centralized point. Types of robots **230** that may be managed include, but are not limited to, attended robots **232**, unattended robots **234**, development robots (similar to unattended robots **234**, but used for development and testing purposes), and nonproduction robots (similar to attended robots **232**, but used for development and testing purposes). Attended robots **232** are triggered by user events and operate alongside a human on the same computing system. Attended robots **232** may be used with conductor **220** for a centralized process deployment and logging medium. Attended robots

232 may help the human user accomplish various tasks, and may be triggered by user events. In some embodiments, processes cannot be started from conductor **220** on this type of robot and/or they cannot run under a locked screen. In certain embodiments, attended robots **232** can only be started from a robot tray or from a command prompt. Attended robots **232** should run under human supervision in some embodiments.

Unattended robots **234** run unattended in virtual environments and can automate many processes. Unattended robots **234** may be responsible for remote execution, monitoring, scheduling, and providing support for work queues. Debugging for all robot types may be run in designer **210** in some embodiments. Both attended and unattended robots may automate (as represented by dashed box **290**) various systems and applications including, but not limited to, mainframes, web applications, VMs, enterprise applications (e.g., those produced by SAP®, SalesForce®, Oracle®, etc.), and computing system applications (e.g., desktop and laptop applications, mobile device applications, wearable computer applications, etc.).

Conductor **220** may have various capabilities (as represented by arrow **232**) including, but not limited to, provisioning, deployment, configuration, queueing, monitoring, logging, and/or providing interconnectivity. Provisioning may include creating and maintenance of connections between robots **230** and conductor **220** (e.g., a web application). Deployment may include assuring the correct delivery of package versions to assigned robots **230** for execution. Configuration may include maintenance and delivery of robot environments and process configurations. Queueing may include providing management of queues and queue items. Monitoring may include keeping track of robot identification data and maintaining user permissions. Logging may include storing and indexing logs to a database (e.g., a structured query language (SQL) or NoSQL database) and/or another storage mechanism (e.g., ElasticSearch®, which provides the ability to store and quickly query large datasets). Conductor **220** may provide interconnectivity by acting as the centralized point of communication for third-party solutions and/or applications.

Robots **230** are execution agents that implement workflows built in designer **210**. One commercial example of some embodiments of robot(s) **230** is UiPath Robots™. In some embodiments, robots **230** install the Microsoft Windows® Service Control Manager (SCM)-managed service by default. As a result, such robots **230** can open interactive Windows® sessions under the local system account, and have the rights of a Windows® service.

In some embodiments, robots **230** can be installed in a user mode. For such robots **230**, this means they have the same rights as the user under which a given robot **230** has been installed. This feature may also be available for High Density (HD) robots, which ensure full utilization of each machine at its maximum potential. In some embodiments, any type of robot **230** may be configured in an HD environment.

Robots **230** in some embodiments are split into several components, each being dedicated to a particular automation task. The robot components in some embodiments include, but are not limited to, SCM-managed robot services, user mode robot services, executors, agents, and command line. SCM-managed robot services manage and monitor Windows® sessions and act as a proxy between conductor **220** and the execution hosts (i.e., the computing systems on which robots **230** are executed). These services are trusted with and manage the credentials for robots **230**. A console application is launched by the SCM under the local system.

User mode robot services in some embodiments manage and monitor Windows® sessions and act as a proxy between conductor **220** and the execution hosts. User mode robot services may be trusted with and manage the credentials for robots **230**. A Windows® application may automatically be launched if the SCM-managed robot service is not installed.

Executors may run given jobs under a Windows® session (i.e., they may execute workflows. Executors may be aware of per-monitor dots per inch (DPI) settings. Agents may be Windows® Presentation Foundation (WPF) applications that display the available jobs in the system tray window. Agents may be a client of the service. Agents may request to start or stop jobs and change settings. The command line is a client of the service. The command line is a console application that can request to start jobs and waits for their output.

Having components of robots **230** split as explained above helps developers, support users, and computing systems more easily run, identify, and track what each component is executing. Special behaviors may be configured per component this way, such as setting up different firewall rules for the executor and the service. The executor may always be aware of DPI settings per monitor in some embodiments. As a result, workflows may be executed at any DPI, regardless of the configuration of the computing system on which they were created. Projects from designer **210** may also be independent of browser zoom level in some embodiments. For applications that are DPI-unaware or intentionally marked as unaware, DPI may be disabled in some embodiments.

RPA system **200** in this embodiment is part of a hyper-automation system. Developers may use designer **210** to build and test RPA robots that utilize AI/ML models deployed in core hyper-automation system **240** (e.g., as part of an AI center thereof). Such RPA robots may send input for execution of the AI/ML model(s) and receive output therefrom via core hyper-automation system **240**.

One or more of robots **230** may be listeners, as described above. These listeners may provide information to core hyper-automation system **240** regarding what users are doing when they use their computing systems. This information may then be used by core hyper-automation system for process mining, task mining, task capture, etc.

An assistant/chatbot **250** may be provided on user computing systems to allow users to launch RPA local robots. The assistant may be located in a system tray, for example. Chatbots may have a user interface so users can see text in the chatbot. Alternatively, chatbots may lack a user interface and run in the background, listening using the computing system's microphone for user speech.

In some embodiments, data labeling may be performed by a user of the computing system on which a robot is executing or on another computing system that the robot provides information to. For instance, if a robot calls an AI/ML model that performs CV on images for VM users, but the AI/ML model does not correctly identify a button on the screen, the user may draw a rectangle around the misidentified or non-identified component and potentially provide text with a correct identification. This information may be provided to core hyper-automation system **240** and then used later for training a new version of the AI/ML model.

FIG. **3** is an architectural diagram illustrating a deployed RPA system **300**, according to one or more embodiments. In some embodiments, RPA system **300** may be a part of RPA system **200** of FIG. **2** and/or hyper-automation system **100** of FIG. **1**. Deployed RPA system **300** may be a cloud-based system, an on-premises system, a desktop-based system that offers enterprise level, user level, or device level automation solutions for automation of different computing processes, etc.

It should be noted that a client side **301**, a server side **302**, or both, may include any desired number of computing systems without deviating from the scope of the one or more embodiments herein. On the client side **301**, a robot application **310** includes executors **312**, an agent **314**, and a designer **316**. However, in some embodiments, designer **316** may not be running on the same computing system as executors **312** and agent **314**. Executors **312** are running processes. Several business projects may run simultaneously, as shown in FIG. **3**. Agent **314** (e.g., a Windows® service) is the single point of contact for all executors **312** in this embodiment. All messages in this embodiment are logged into conductor **340**, which processes them further via database server **355**, an AI/ML server **360**, an indexer server **370**, or any combination thereof. As discussed above with respect to FIG. **2**, executors **312** may be robot components.

In some embodiments, a robot represents an association between a machine name and a username. The robot may manage multiple executors at the same time. On computing systems that support multiple interactive sessions running simultaneously (e.g., Windows® Server 2012), multiple robots may be running at the same time, each in a separate Windows® session using a unique username. This is referred to as HD robots above.

Agent **314** is also responsible for sending the status of the robot (e.g., periodically sending a "heartbeat" message indicating that the robot is still functioning) and downloading the required version of the package to be executed. The communication between agent **314** and conductor **340** is always initiated by agent **314** in some embodiments. In the notification scenario, agent **314** may open a WebSocket channel that is later used by conductor **330** to send commands to the robot (e.g., start, stop, etc.).

A listener **330** monitors and records data pertaining to user interactions with an attended computing system and/or operations of an unattended computing system on which listener **330** resides. Listener **330** may be an RPA robot, part of an operating system, a downloadable application for the respective computing system, or any other software and/or hardware without deviating from the scope of the one or more embodiments herein. Indeed, in some embodiments, the logic of the listener is implemented partially or completely via physical hardware.

On the server side **302**, a presentation layer **333**, a service layer **334**, and a persistence layer **336** are included, as well as a conductor **340**. The presentation layer **333** can include a web application **342**, Open Data Protocol (OData) Representative State Transfer (REST) Application Programming Interface (API) endpoints **344**, and notification and monitoring **346**. The service layer **334** can include API implementation/business logic **348**. The persistence layer **336** can include a database server **355**, an AI/ML server **360**, and an indexer server **370**. For example, the conductor **340** includes the web application **342**, the OData REST API endpoints **344**, the notification and monitoring **346**, and the API implementation/business logic **348**. In some embodiments, most actions that a user performs in the interface of the conductor **340** (e.g., via browser **320**) are performed by calling various APIs. Such actions may include, but are not limited to, starting jobs on robots, adding/removing data in queues, scheduling jobs to run unattended, etc. without deviating from the scope of the one or more embodiments

herein. The web application **342** can be the visual layer of the server platform. In this embodiment, the web application **342** uses Hypertext Markup Language (HTML) and JavaScript (JS). However, any desired markup languages, script languages, or any other formats may be used without deviating from the scope of the one or more embodiments herein. The user interacts with web pages from the web application **342** via the browser **320** in this embodiment in order to perform various actions to control conductor **340**. For instance, the user may create robot groups, assign packages to the robots, analyze logs per robot and/or per process, start and stop robots, etc.

In addition to web application **342**, conductor **340** also includes service layer **334** that exposes OData REST API endpoints **344**. However, other endpoints may be included without deviating from the scope of the one or more embodiments herein. The REST API is consumed by both web application **342** and agent **314**. Agent **314** is the supervisor of one or more robots on the client computer in this embodiment.

The REST API in this embodiment includes configuration, logging, monitoring, and queueing functionality (represented by at least arrow **349**). The configuration endpoints may be used to define and configure application users, permissions, robots, assets, releases, and environments in some embodiments. Logging REST endpoints may be used to log different information, such as errors, explicit messages sent by the robots, and other environment-specific information, for instance. Deployment REST endpoints may be used by the robots to query the package version that should be executed if the start job command is used in conductor **340**. Queueing REST endpoints may be responsible for queues and queue item management, such as adding data to a queue, obtaining a transaction from the queue, setting the status of a transaction, etc.

Monitoring REST endpoints may monitor web application **342** and agent **314**. Notification and monitoring API **346** may be REST endpoints that are used for registering agent **314**, delivering configuration settings to agent **314**, and for sending/receiving notifications from the server and agent **314**. Notification and monitoring API **346** may also use WebSocket communication in some embodiments. As shown in FIG. **3**, one or more the activities/actions described herein are represented by arrows **350** and **351**.

The APIs in the service layer **334** may be accessed through configuration of an appropriate API access path in some embodiments, e.g., based on whether conductor **340** and an overall hyper-automation system have an on-premises deployment type or a cloud-based deployment type. APIs for conductor **340** may provide custom methods for querying stats about various entities registered in conductor **340**. Each logical resource may be an OData entity in some embodiments. In such an entity, components such as the robot, process, queue, etc., may have properties, relationships, and operations. APIs of conductor **340** may be consumed by web application **342** and/or agents **314** in two ways in some embodiments: by getting the API access information from conductor **340**, or by registering an external application to use the OAuth flow.

The persistence layer **336** includes a trio of servers in this embodiment—database server **355** (e.g., a SQL server), AI/ML server **360** (e.g., a server providing AI/ML model serving services, such as AI center functionality) and indexer server **370**. Database server **355** in this embodiment stores the configurations of the robots, robot groups, associated processes, users, roles, schedules, etc. This information is managed through web application **342** in some embodiments. Database server **355** may manage queues and queue items. In some embodiments, database server **355** may store messages logged by the robots (in addition to or in lieu of indexer server **370**). Database server **355** may also store process mining, task mining, and/or task capture-related data, received from listener **330** installed on the client side **301**, for example. While no arrow is shown between listener **330** and database **355**, it should be understood that listener **330** is able to communicate with database **355**, and vice versa in some embodiments. This data may be stored in the form of PDDs, images, XAML files, etc. Listener **330** may be configured to intercept user actions, processes, tasks, and performance metrics on the respective computing system on which listener **330** resides. For example, listener **330** may record user actions (e.g., clicks, typed characters, locations, applications, active elements, times, etc.) on its respective computing system and then convert these into a suitable format to be provided to and stored in database server **355**.

AI/ML server **360** facilitates incorporation of AI/ML models into automations. Pre-built AI/ML models, model templates, and various deployment options may make such functionality accessible even to those who are not data scientists. Deployed automations (e.g., RPA robots) may call AI/ML models from AI/ML server **360**. Performance of the AI/ML models may be monitored, and be trained and improved using human-validated data. AI/ML server **360** may schedule and execute training jobs to train new versions of the AI/ML models.

AI/ML server **360** may store data pertaining to AI/ML models and ML packages for configuring various ML skills for a user at development time. An ML skill, as used herein, is a pre-built and trained ML model for a process, which may be used by an automation, for example. AI/ML server **460** may also store data pertaining to document understanding technologies and frameworks, algorithms and software packages for various AI/ML capabilities including, but not limited to, intent analysis, natural language processing (NLP), speech analysis, different types of AI/ML models, etc.

Indexer server **370**, which is optional in some embodiments, stores and indexes the information logged by the robots. In certain embodiments, indexer server **370** may be disabled through configuration settings. In some embodiments, indexer server **370** uses ElasticSearch®, which is an open source project full-text search engine. Messages logged by robots (e.g., using activities like log message or write line) may be sent through the logging REST endpoint(s) to indexer server **370**, where they are indexed for future utilization.

FIG. **4** is an architectural diagram illustrating the relationship between a designer **410**, activities **420**, **430**, **440**, **450**, drivers **460**, APIs **470**, and AI/ML models **480** according to one or more embodiments. As described herein, a developer uses the designer **410** to develop workflows that are executed by robots. The various types of activities may be displayed to the developer in some embodiments. Designer **410** may be local to the user's computing system or remote thereto (e.g., accessed via VM or a local web browser interacting with a remote web server). Workflows may include user-defined activities **420**, API-driven activities **430**, AI/ML activities **440**, and/or UI automation activities **450**. By way of example (as shown by the dotted lines), user-defined activities **420** and API-driven activities **440** interact with applications via their APIs. In turn, User-defined activities **420** and/or AI/ML activities **440** may call one or more AI/ML models **480** in some embodiments,

which may be located locally to the computing system on which the robot is operating and/or remotely thereto.

Some embodiments are able to identify non-textual visual components in an image, which is called CV herein. CV may be performed at least in part by AI/ML model(s) **480**. Some CV activities pertaining to such components may include, but are not limited to, extracting of text from segmented label data using OCR, fuzzy text matching, cropping of segmented label data using ML, comparison of extracted text in label data with ground truth data, etc. In some embodiments, there may be hundreds or even thousands of activities that may be implemented in user-defined activities **420**. However, any number and/or type of activities may be used without deviating from the scope of the one or more embodiments herein.

UI automation activities **450** are a subset of special, lower-level activities that are written in lower-level code and facilitate interactions with the screen. UI automation activities **450** facilitate these interactions via drivers **460** that allow the robot to interact with the desired software. For instance, drivers **460** may include operating system (OS) drivers **462**, browser drivers **464**, VM drivers **466**, enterprise application drivers **468**, etc. One or more of AI/ML models **480** may be used by UI automation activities **450** in order to perform interactions with the computing system in some embodiments. In certain embodiments, AI/ML models **480** may augment drivers **460** or replace them completely. Indeed, in certain embodiments, drivers **460** are not included.

Drivers **460** may interact with the OS at a low level looking for hooks, monitoring for keys, etc. via OS drivers **462**. Drivers **460** may facilitate integration with Chrome®, IE®, Citrix®, SAP®, etc. For instance, the "click" activity performs the same role in these different applications via drivers **460**.

FIG. **5** is an architectural diagram illustrating a computing system **500** configured to provide type cache for package management of RPAs according to one or more embodiments. In some embodiments, computing system **500** may be one or more of the computing systems depicted and/or described herein. In certain embodiments, computing system **500** may be part of a hyper-automation system, such as that shown in FIGS. **1** and **2**. Computing system **500** includes a bus **505** or other communication mechanism for communicating information, and processor(s) **510** coupled to bus **505** for processing information. Processor(s) **510** may be any type of general or specific purpose processor, including a Central Processing Unit (CPU), an Application Specific Integrated Circuit (ASIC), a Field Programmable Gate Array (FPGA), a Graphics Processing Unit (GPU), multiple instances thereof, and/or any combination thereof. Processor(s) **510** may also have multiple processing cores, and at least some of the cores may be configured to perform specific functions. Multi-parallel processing may be used in some embodiments. In certain embodiments, at least one of processor(s) **510** may be a neuromorphic circuit that includes processing elements that mimic biological neurons. In some embodiments, neuromorphic circuits may not require the typical components of a Von Neumann computing architecture.

Computing system **500** further includes a memory **515** for storing information and instructions to be executed by processor(s) **510**. Memory **515** can be comprised of any combination of random access memory (RAM), read-only memory (ROM), flash memory, cache, static storage such as a magnetic or optical disk, or any other types of non-transitory computer-readable media or combinations thereof. Non-transitory computer-readable media may be any available media that can be accessed by processor(s) **510** and may include volatile media, non-volatile media, or both. The media may also be removable, non-removable, or both.

Additionally, computing system **500** includes a communication device **520**, such as a transceiver, to provide access to a communications network via a wireless and/or wired connection. In some embodiments, communication device **520** may be configured to use Frequency Division Multiple Access (FDMA), Single Carrier FDMA (SC-FDMA), Time Division Multiple Access (TDMA), Code Division Multiple Access (CDMA), Orthogonal Frequency Division Multiplexing (OFDM), Orthogonal Frequency Division Multiple Access (OFDMA), Global System for Mobile (GSM) communications, General Packet Radio Service (GPRS), Universal Mobile Telecommunications System (UMTS), cdma2000, Wideband CDMA (W-CDMA), High-Speed Downlink Packet Access (HSDPA), High-Speed Uplink Packet Access (HSUPA), High-Speed Packet Access (HSPA), Long Term Evolution (LTE), LTE Advanced (LTE-A), 802.11x, Wi-Fi, Zigbee, Ultra-WideBand (UWB), 802.16x, 802.15, Home Node-B (HnB), Bluetooth, Radio Frequency Identification (RFID), Infrared Data Association (IrDA), Near-Field Communications (NFC), fifth generation (5G) New Radio (NR), any combination thereof, and/or any other currently existing or future-implemented communications standard and/or protocol without deviating from the scope of the one or more embodiments herein. In some embodiments, communication device **520** may include one or more antennas that are singular, arrayed, panels, phased, switched, beamforming, beamsteering, a combination thereof, and or any other antenna configuration without deviating from the scope of the one or more embodiments herein.

Processor(s) **510** are further coupled via bus **505** to a display **525**, such as a plasma display, a Liquid Crystal Display (LCD), a Light Emitting Diode (LED) display, a Field Emission Display (FED), an Organic Light Emitting Diode (OLED) display, a flexible OLED display, a flexible substrate display, a projection display, a **4**K display, a high definition display, a Retina® display, an In-Plane Switching (IPS) display, or any other suitable display for displaying information to a user. Display **525** may be configured as a touch (haptic) display, a three-dimensional (3D) touch display, a multi-input touch display, a multi-touch display, etc. using resistive, capacitive, surface-acoustic wave (SAW) capacitive, infrared, optical imaging, dispersive signal technology, acoustic pulse recognition, frustrated total internal reflection, etc. Any suitable display device and haptic I/O may be used without deviating from the scope of the one or more embodiments herein.

A keyboard **530** and a cursor control device **535**, such as a computer mouse, a touchpad, etc., are further coupled to bus **505** to enable a user to interface with computing system **500**. However, in certain embodiments, a physical keyboard and mouse may not be present, and the user may interact with the device solely through display **525** and/or a touchpad (not shown). Any type and combination of input devices may be used as a matter of design choice. In certain embodiments, no physical input device and/or display is present. For instance, the user may interact with computing system **500** remotely via another computing system in communication therewith, or computing system **500** may operate autonomously.

Memory **515** stores software modules that provide functionality when executed by processor(s) **510**. The modules

include an operating system **540** for computing system **500**. The modules further include a module **545** (such as a type cache module implementing a type cache service) that is configured to perform all or part of the processes described herein or derivatives thereof. Computing system **500** may include one or more additional functional modules **550** that include additional functionality.

One skilled in the art will appreciate that a "system" could be embodied as a server, an embedded computing system, a personal computer, a console, a personal digital assistant (PDA), a cell phone, a tablet computing device, a quantum computing system, or any other suitable computing device, or combination of devices without deviating from the scope of the one or more embodiments herein. Presenting the above-described functions as being performed by a "system" is not intended to limit the scope of embodiments herein in any way, but is intended to provide one example of the many embodiments. Indeed, methods, systems, and apparatuses disclosed herein may be implemented in localized and distributed forms consistent with computing technology, including cloud computing systems. The computing system could be part of or otherwise accessible by a local area network (LAN), a mobile communications network, a satellite communications network, the Internet, a public or private cloud, a hybrid cloud, a server farm, any combination thereof, etc. Any localized or distributed architecture may be used without deviating from the scope of the one or more embodiments herein.

It should be noted that some of the system features described in this specification have been presented as modules, in order to more particularly emphasize their implementation independence. For example, a module may be implemented as a hardware circuit comprising custom very large scale integration (VLSI) circuits or gate arrays, off-the-shelf semiconductors such as logic chips, transistors, or other discrete components. A module may also be implemented in programmable hardware devices such as field programmable gate arrays, programmable array logic, programmable logic devices, graphics processing units, or the like.

A module may also be at least partially implemented in software for execution by various types of processors. An identified unit of executable code may, for instance, include one or more physical or logical blocks of computer instructions that may, for instance, be organized as an object, procedure, or function. Nevertheless, the executables of an identified module need not be physically located together, but may include disparate instructions stored in different locations that, when joined logically together, comprise the module and achieve the stated purpose for the module. Further, modules may be stored on a computer-readable medium, which may be, for instance, a hard disk drive, flash device, RAM, tape, and/or any other such non-transitory computer-readable medium used to store data without deviating from the scope of the one or more embodiments herein.

Indeed, a module of executable code could be a single instruction, or many instructions, and may even be distributed over several different code segments, among different programs, and across several memory devices. Similarly, operational data may be identified and illustrated herein within modules, and may be embodied in any suitable form and organized within any suitable type of data structure. The operational data may be collected as a single data set, or may be distributed over different locations including over different storage devices, and may exist, at least partially, merely as electronic signals on a system or network.

Various types of AI/ML models may be trained and deployed without deviating from the scope of the one or more embodiments herein. For instance, FIG. **6** illustrates an example of a neural network **600** that has been trained to recognize graphical elements in an image according to one or more embodiments. Here, neural network **600** receives pixels (as represented by column **610**) of a screenshot image of a 1920×1080 screen as input for input "neurons" 1 to I of an input layer (as represented by column **620**). In this case, I is 2,073,600, which is the total number of pixels in the screenshot image.

Neural network **600** also includes a number of hidden layers (as represented by column **630** and **640**). Both DLNNs and shallow learning neural networks (SLNNs) usually have multiple layers, although SLNNs may only have one or two layers in some cases, and normally fewer than DLNNs. Typically, the neural network architecture includes the input layer, multiple intermediate layers (e.g., the hidden layers), and an output layer (as represented by column **650**), as is the case in neural network **600**.

A DLNN often has many layers (e.g., 10, 50, 200, etc.) and subsequent layers typically reuse features from previous layers to compute more complex, general functions. A SLNN, on the other hand, tends to have only a few layers and train relatively quickly since expert features are created from raw data samples in advance. However, feature extraction is laborious. DLNNs, on the other hand, usually do not require expert features, but tend to take longer to train and have more layers.

For both approaches, the layers are trained simultaneously on the training set, normally checking for overfitting on an isolated cross-validation set. Both techniques can yield excellent results, and there is considerable enthusiasm for both approaches. The optimal size, shape, and quantity of individual layers varies depending on the problem that is addressed by the respective neural network.

Returning to FIG. **6**, pixels provided as the input layer are fed as inputs to the J neurons of hidden layer 1. While all pixels are fed to each neuron in this example, various architectures are possible that may be used individually or in combination including, but not limited to, feed forward networks, radial basis networks, deep feed forward networks, deep convolutional inverse graphics networks, convolutional neural networks, recurrent neural networks, artificial neural networks, long/short term memory networks, gated recurrent unit networks, generative adversarial networks, liquid state machines, auto encoders, variational auto encoders, denoising auto encoders, sparse auto encoders, extreme learning machines, echo state networks, Markov chains, Hopfield networks, Boltzmann machines, restricted Boltzmann machines, deep residual networks, Kohonen networks, deep belief networks, deep convolutional networks, support vector machines, neural Turing machines, or any other suitable type or combination of neural networks without deviating from the scope of the one or more embodiments herein.

Hidden layer 2 (**630**) receives inputs from hidden layer 1 (**620**), hidden layer 3 receives inputs from hidden layer 2 (**630**), and so on for all hidden layers until the last hidden layer (as represented by the ellipses **655**) provides its outputs as inputs for the output layer. It should be noted that numbers of neurons I, J, K, and L are not necessarily equal, and thus, any desired number of layers may be used for a given layer of neural network **600** without deviating from the scope of the one or more embodiments herein. Indeed, in certain embodiments, the types of neurons in a given layer may not all be the same.

Neural network **600** is trained to assign a confidence score to graphical elements believed to have been found in the image. In order to reduce matches with unacceptably low likelihoods, only those results with a confidence score that meets or exceeds a confidence threshold may be provided in some embodiments. For instance, if the confidence threshold is 80%, outputs with confidence scores exceeding this amount may be used and the rest may be ignored. In this case, the output layer indicates that two text fields (as represented by outputs **661** and **662**), a text label (as represented by output **663**), and a submit button (as represented by output **665**) were found. Neural network **600** may provide the locations, dimensions, images, and/or confidence scores for these elements without deviating from the scope of the one or more embodiments herein, which can be used subsequently by an RPA robot or another process that uses this output for a given purpose.

It should be noted that neural networks are probabilistic constructs that typically have a confidence score. This may be a score learned by the AI/ML model based on how often a similar input was correctly identified during training. For instance, text fields often have a rectangular shape and a white background. The neural network may learn to identify graphical elements with these characteristics with a high confidence. Some common types of confidence scores include a decimal number between 0 and 1 (which can be interpreted as a percentage of confidence), a number between negative ∞ and positive ∞, or a set of expressions (e.g., "low," "medium," and "high"). Various post-processing calibration techniques may also be employed in an attempt to obtain a more accurate confidence score, such as temperature scaling, batch normalization, weight decay, negative log likelihood (NLL), etc.

"Neurons" in a neural network are mathematical functions that are typically based on the functioning of a biological neuron. Neurons receive weighted input and have a summation and an activation function that governs whether they pass output to the next layer. This activation function may be a nonlinear thresholded activity function where nothing happens if the value is below a threshold, but then the function linearly responds above the threshold (i.e., a rectified linear unit (ReLU) nonlinearity). Summation functions and ReLU functions are used in deep learning since real neurons can have approximately similar activity functions. Via linear transforms, information can be subtracted, added, etc. In essence, neurons act as gating functions that pass output to the next layer as governed by their underlying mathematical function. In some embodiments, different functions may be used for at least some neurons.

An example of a neuron **700** is shown in FIG. **7**. Inputs x_1, x_2, . . . , x_n from a preceding layer are assigned respective weights w_1, w_2, . . . , w^n. Thus, the collective input from preceding neuron 1 is w_1x_1. These weighted inputs are used for the neuron's summation function modified by a bias, such as:

$$\sum_{i=1}^{m}(w_i x_i) + \text{bias} \tag{1}$$

This summation is compared against an activation function $f(x)$ (as represented by block **710**) to determine whether the neuron "fires". For instance, $f(x)$ may be given by:

$$f(x) = \begin{cases} 1 & \text{if } \sum wx + \text{bias} \geq 0 \\ 0 & \text{if } \sum wx + \text{bias} < 0 \end{cases} \tag{2}$$

The output y of neuron **700** may thus be given by:

$$y = f(x)\sum_{i=1}^{m}(w_i x_i) + \text{bias} \tag{3}$$

In this case, neuron **700** is a single-layer perceptron. However, any suitable neuron type or combination of neuron types may be used without deviating from the scope of the one or more embodiments herein. It should also be noted that the ranges of values of the weights and/or the output value(s) of the activation function may differ in some embodiments without deviating from the scope of the one or more embodiments herein.

The goal, or "reward function" is often employed, such as for this case the successful identification of graphical elements in the image. A reward function explores intermediate transitions and steps with both short-term and long-term rewards to guide the search of a state space and attempt to achieve a goal (e.g., successful identification of graphical elements, successful identification of a next sequence of activities for an RPA workflow, etc.).

During training, various labeled data (in this case, images) are fed through neural network **600**. Successful identifications strengthen weights for inputs to neurons, whereas unsuccessful identifications weaken them. A cost function, such as mean square error (MSE) or gradient descent may be used to punish predictions that are slightly wrong much less than predictions that are very wrong. If the performance of the AI/ML model is not improving after a certain number of training iterations, a data scientist may modify the reward function, provide indications of where non-identified graphical elements are, provide corrections of misidentified graphical elements, etc.

Backpropagation is a technique for optimizing synaptic weights in a feedforward neural network. Backpropagation may be used to "pop the hood" on the hidden layers of the neural network to see how much of the loss every node is responsible for, and subsequently updating the weights in such a way that minimizes the loss by giving the nodes with higher error rates lower weights, and vice versa. In other words, backpropagation allows data scientists to repeatedly adjust the weights so as to minimize the difference between actual output and desired output.

The backpropagation algorithm is mathematically founded in optimization theory. In supervised learning, training data with a known output is passed through the neural network and error is computed with a cost function from known target output, which gives the error for backpropagation. Error is computed at the output, and this error is transformed into corrections for network weights that will minimize the error.

In the case of supervised learning, an example of backpropagation is provided below. A column vector input x is processed through a series of N nonlinear activity functions f_i between each layer i=1, . . . , N of the network, with the output at a given layer first multiplied by a synaptic matrix W_i, and with a bias vector b_1 added. The network output o, given by

$$o=f_N(W_N f_{N-1}(W_{N-1} f_{N-2}(\ldots f_1(W_1 x+b_1) \ldots)+b_{N-1})+b_N) \tag{4}$$

In some embodiments, o is compared with a target output t, resulting in an error

$$E = \frac{1}{2}\|o-t\|^2,$$

which is desired to be minimized.

Optimization in the form of a gradient descent procedure may be used to minimize the error by modifying the synaptic weights W_i for each layer. The gradient descent procedure requires the computation of the output o given an input x corresponding to a known target output t, and producing an error o−t. This global error is then propagated backwards giving local errors for weight updates with computations similar to, but not exactly the same as, those used for forward propagation. In particular, the backpropagation step typically requires an activity function of the form $p_j(n_j)=f_j'(n_j)$, where n_j is the network activity at layer j (i.e., $n_j=W_j o_{j-1}+b_j$) where $o_j=f_j(n_j)$ and the apostrophe ' denotes the derivative of the activity function f.

The weight updates may be computed via the formulae:

$$d_j = \begin{cases} (o-t)\circ p_j(n_j), & j=N \\ W_{j+1}^T d_{j+1}\circ p_j(n_j), & j<N \end{cases} \tag{5}$$

$$\frac{\partial E}{\partial W_{j+1}} = d_{j+1}(o_j)^T \tag{6}$$

$$\frac{\partial E}{\partial b_{j+1}} = d_{j+1} \tag{7}$$

$$W_j^{new} = W_j^{old} - \eta\frac{\partial E}{\partial W_j} \tag{8}$$

$$b_j^{new} = b_j^{old} - \eta\frac{\partial E}{\partial b_j} \tag{9}$$

where $\circ$ denotes a Hadamard product (i.e., the element-wise product of two vectors), T denotes the matrix transpose, and o_j denotes $f_j(W_j o_{j-1}+b_j)$, with $o_0=x$. Here, the learning rate η is chosen with respect to machine learning considerations. Below, η is related to the neural Hebbian learning mechanism used in the neural implementation. Note that the synapses Wand b can be combined into one large synaptic matrix, where it is assumed that the input vector has appended ones, and extra columns representing the b synapses are subsumed to W.

The AI/ML model may be trained over multiple epochs until it reaches a good level of accuracy (e.g., 97% or better using an F2 or F4 threshold for detection and approximately 2,000 epochs). This accuracy level may be determined in some embodiments using an F1 score, an F2 score, an F4 score, or any other suitable technique without deviating from the scope of the one or more embodiments herein. Once trained on the training data, the AI/ML model may be tested on a set of evaluation data that the AI/ML model has not encountered before. This helps to ensure that the AI/ML model is not "over fit" such that it identifies graphical elements in the training data well, but does not generalize well to other images.

In some embodiments, it may not be known what accuracy level is possible for the AI/ML model to achieve. Accordingly, if the accuracy of the AI/ML model is starting to drop when analyzing the evaluation data (i.e., the model is performing well on the training data, but is starting to perform less well on the evaluation data), the AI/ML model may go through more epochs of training on the training data (and/or new training data). In some embodiments, the AI/ML model is only deployed if the accuracy reaches a certain level or if the accuracy of the trained AI/ML model is superior to an existing deployed AI/ML model.

In certain embodiments, a collection of trained AI/ML models may be used to accomplish a task, such as employing an AI/ML model for each type of graphical element of interest, employing an AI/ML model to perform OCR, deploying yet another AI/ML model to recognize proximity relationships between graphical elements, employing still another AI/ML model to generate an RPA workflow based on the outputs from the other AI/ML models, etc. This may collectively allow the AI/ML models to enable semantic automation, for instance.

Some embodiments may use transformer networks such as SentenceTransformers™, which is a Python™ framework for state-of-the-art sentence, text, and image embeddings. Such transformer networks learn associations of words and phrases that have both high scores and low scores. This trains the AI/ML model to determine what is close to the input and what is not, respectively. Rather than just using pairs of words/phrases, transformer networks may use the field length and field type, as well.

FIG. **8** is a flowchart illustrating a process **800** for training AI/ML model(s) according to one or more embodiments. Note that the process **800** can also be applied to other UI learning operations, such as for NLP and chatbots. The process begins with training data, for example providing labeled data as illustrated in FIG. **8**, such as labeled screens (e.g., with graphical elements and text identified), words and phrases, a "thesaurus" of semantic associations between words and phrases such that similar words and phrases for a given word or phrase can be identified, etc. at block **810**. The nature of the training data that is provided will depend on the objective that the AI/ML model is intended to achieve. The AI/ML model is then trained over multiple epochs at block **820** and results are reviewed at block **830**.

If the AI/ML model fails to meet a desired confidence threshold at decision block **840** (the process **800** proceeds according to the NO arrow), the training data is supplemented and/or the reward function is modified to help the AI/ML model achieve its objectives better at block **850** and the process returns to block **820**. If the AI/ML model meets the confidence threshold at decision block **840** (the process **800** proceeds according to the YES arrow), the AI/ML model is tested on evaluation data at block **860** to ensure that the AI/ML model generalizes well and that the AI/ML model is not over fit with respect to the training data. The evaluation data may include screens, source data, etc. that the AI/ML model has not processed before. If the confidence threshold is met at decision block **870** for the evaluation data (the process **800** proceeds according to the Yes arrow), the AI/ML model is deployed at block **880**. If not (the process **800** proceeds according to the NO arrow), the process returns to block **880** and the AI/ML model is trained further.

FIG. **9** is a flowchart illustrating a process **900** for providing type cache for package management of RPAs according to one or more embodiments. The process **800** performed in FIG. **8** and the process **900** performed in FIG. **9** may be performed by a computer program, encoding instructions for the processor(s), in accordance with one or more embodiments. The computer program may be embodied on a non-transitory computer-readable medium. The computer-readable medium may be, but is not limited to, a hard disk drive, a flash device, RAM, a tape, and/or any other such medium or combination of media used to store data. The computer program may include encoded instructions for controlling processor(s) of a computing system (e.g., processor(s) **510** of computing system **500** of FIG. **5**) to implement all or part of the process steps described in FIGS. **8-9**, which may also be stored on the computer-readable medium.

Generally, the process **900** can be considered a type cache service that operates/works in background by proactively scanning all feeds that are subject to scanning and automatically detecting if there is something new on the feed. Proactively scanning and automatically include, but are not limited to, detecting differences in packages, detecting differences in package behavior over time, automating a processing of scan and determination results, improving reliability of the feed. A feed can include, but is not limited to, a stream of publication of packages that the process **900** discover upon scanning. The feed can be identified by a name, a source URL, an access token, etc. For instance, if a new Microsoft Office package is published, the new Microsoft Office package is automatically scanned by the type cache service in the background while the new Microsoft Office package is still in an unpacked stage (e.g., an unpacked NuGet). The scanning by the type cache service discovers all assemblies inside the unpacked package, along with metadata and JSON file attached to the unpacked package. The scanning is performed by the type cache service to detect which activities are published inside the unpacked package. All the detected activities are stored in a database (i.e., a cache) with indexing. Further, this cache data is used to provide information on each available activity in the unpacked package upon a search without a NuGet Restoration and make the activities discoverable.

In operation, the process **900** begins at block **910**. At block **910**, a computing system publishes one or more packages. Each package can include one or more activities. For example, a package can be a NuGet package with a list of assemblies that have types/objects defined therein. When the Nuget package is published, the computing system publishes the NuGet package to a feed (e.g., a web server or Nuget server). The computing system, to implement an effective type cache service, seeks to discover names, versions, and dependencies to answer 'what are the types (e.g., activities that we can use)?' and 'what are the dependencies?'.

At block **930**, the type cache service of the computing system scans the feed. The type cache service can also scan all available folders. These one or more scans can be executed periodically, on a time scheduled, and/or triggered. During these one or more scans, the type cache service answers 'what are the types (e.g., activities that we can use)?' and 'what are the dependencies?' by discovering names and version, as well as determining types/activities/.net objects/etc. According to one or more embodiments, the type cache service resolves dependencies by building dependency tree and implementing conflict resolutions across the information of the feed, folders, and existing information. The dependency tree can be a directed graph representing dependencies of several objects towards each other, and can be derived into an evaluation order or the absence of an evaluation order that respects the given dependencies from the dependency graph.

At block **940**, the type cache service of the computing system indexes all activities (e.g., new activities and all previously detected activities) according to type. In an example use case, metadata (e.g., a activity name, file name, user name) about the types can be used for indexing. Further, a package owner can be leveraged for confirmation of metadata, as well as use of a translation key.

According to one or more embodiments, the computing system can utilize a package management software where a developer adds references to one or more packages or versions thereof and uses those references (and packages) to build a workflow. Turning to FIG. **10**, an interface **1000** is depicted according to one or more embodiments. The interface **1000** is an example of adding an activity to a workflow canvas in the RPA studio web software, where the interface **1000** includes at least an activity **1010** that is further configurable upon clicking and an add icon **1015** that is also clickable to enable adding further activities. Note that the type cache service builds (i.e., indexes in view thereof) the type cache on top of those packages to maintain a friendly user experience within the package management software.

At block **950**, the type cache service of the computing system receives one or more inputs. The one or more inputs are utilized to search the indexes (i.e., the type cache). One or more technical, effects, and benefits of the type cache service include enabling the searching of the indexes directly. FIG. **11** depicts an interface **1100** according to one or more embodiments. The interface **1100** provides a search bar **1110** to search activities. Based on the one or more inputs, the activities category header **1115** is followed by one or more types **1120**, **1125**, **1130**, **1135**, **1140**, **1145**, and **1150**. Each of the one or more types **1120**, **1125**, **1130**, **1135**, **1140**, **1145**, and **1150** are clickable to further discover all activities listed under that corresponding type. For example, the type **1120** can be default activity category, the type **1125** can be UiPath MicrosoftOffice360 category, the type **1130** can be UiPath system activity category, the type **1135** can be UiPath UiAutomation activity category, the type **1140** can be UiPath WebAFI activity category, the type **1145** can be UiPath AmazonWebService activity category, and the type **1145** can be UiPath AmazonWorkSpaces activity category.

By way of example and as discussed herein, performing a NuGet restoration is slow because the NuGet restoration queries different services multiple times to construct a dependency graph. In some cases, a NuGet framework attempt to leverage a local storage cache to save all known packages for querying, which marginally improves speed on subsequent queries while initial queries remain entirely slow. Further, in a web environment, the NuGet framework does not provide access to any local storage, which means all queries remain slow. In the NuGet framework, there is no mechanism that can meet a search need of a designer that needs to find a package installer to use an activity in the workflow. Further, evening if the package is found or known, an activity required for use in the workflow needs to be searched within the package. In most cases, the designer should have the knowledge of which activity consists of what package and package version before searching for an activity. In contrast, an RPA studio web software (such as UiPath Studio™) that leverages the type cache service improves the search process. FIG. **12** depicts an interface **1200** according to one or more embodiments. The interface **1200** provides a search bar **1210** to search activities. Upon receiving one or more inputs (e.g., the designer type in 'get newest'), the type cache service automatically display a result number **1220** (e.g., "2 activities found"), as well as activities **1240** and **1250**. For example, the activity **1240** can be a 'get newest email' activity with respect to Microsoft Outlook that returns the most recent email that matches the search criteria. Further, the activity **1250** can be a 'get newest email' activity with respect to Google Mail that returns the most recent email that matches the search criteria. For instance, when the designer searches for an activity type in the search bat **1210** (e.g. "get newest email" activity), the type cache service reads all the latest emails. Then, on typing the 'Latest email', the packages are automatically scanned to select the latest version of the activity and starts installing in the background along with all the references of the package.

At block **970**, the type cache service of the computing system populates a subset of activities matching the search (i.e., the one or more inputs). At block **980**, the type cache service of the computing system receives a selection of the populated subset of activities. The selection causes a restoration operation of the corresponding activity.

FIG. **13** depicts an architectural diagram illustrating a computing environment **1300** according to one or more embodiments. The computing environment **1300** includes storage **1310** (e.g., Azure Storage). The storage **1310** can further include one or more folder feeds **1311**, **1313**, **1315**, and **1317**. The computing environment **1300** also includes a type cache service **1310**. The type cache service **1310** can further include a resolver **1331** (e.g., NuGet Cache Resolver), a client **1333** (e.g., NuGet Client), a package scanner **1335**, and an assembly scanner **1337**. The computing environment **1300** also includes a type database **1340**.

In operation, a designer can initiate or open a project, such as in designer application (UiPath Studio™) to create and deploy automations more rapidly. A project contains a list of package dependencies that are needed for activities used in a workflow. When a project is opened, the type cache service **1310** is queried for a list of DLLs needed to run that workflow. In turn, the type cache service **1310** can quickly determine a dependency graph because all packages have been previously indexed. Further, selection of a package or an activity can cause a restore operation to be triggered.

Next, the resolver **1331** prepares the type cache, such as within the type database **1340** to provide visibility of the packages and tailor user experience. That is, since the designer has limited visibility to the one or more folder feeds **1311**, **1313**, **1315**, and **1317** of the storage **1310**, these folder feeds **1311**, **1313**, **1315**, and **1317** are included in this restore operation. For example, type cache is generated as the type cache service **1310** walks the one or more folder feeds **1311**, **1313**, **1315**, and **1317** and proactively extracts info on a routine schedule to index the packages (e.g., so the packages are ready for use).

The designer can also upload the project, which means that there may be dependencies in that project that the type cache has not seen before. New dependencies can cause the type cache service **1310** to perform an install-scan-index cycle (by using the package scanner **1335** and/or the assembly scanner **1337**).

A collection of type cache (e.g., activities and intelligence data) is passed back to the designer. If the DLLs come from the client **1333**, this corresponding type cache can come from the type database **1340**. If DLLs are downloaded they are scanned by the package scanner **1335** and/or the assembly scanner **1337**, the type database **1340** is updated and the type cache sent back.

One or more technical effects, advantages, or benefits of the type cache service **1310** include enabling the designer to directly search for activities within unpacked packages and resolving a discoverability problem for the unpacked packages. Further, the type cache service **1310** eliminates or bypasses manual installation of each package.

FIG. **14** is a flowchart illustrating a process **1400** according to one or more embodiments. The process **1400** performed in FIG. **14** may be performed by a computer program, encoding instructions for the processor(s), in accordance with one or more embodiments. The computer program may be embodied on a non-transitory computer-readable medium. The computer-readable medium may be, but is not limited to, a hard disk drive, a flash device, RAM, a tape, and/or any other such medium or combination of media used to store data. The computer program may include encoded instructions for controlling processor(s) of a computing system (e.g., processor(s) **510** of computing system **500** of FIG. **5**) to implement all or part of the process steps described in FIGS. **8-9** and **14**, which may also be stored on the computer-readable medium.

Generally, the process **1400** can be considered a type cache service to provide information on each available activity in the one or more unpacked package upon a search without a restoration operation. The type cache service operates/works in background by proactively scanning all feeds that are subject to scanning and automatically detecting if there is something new on the feed. In conjunction with scanning and detecting, the process **1400** includes dependency resolution operations with respect to a dependency graph (or the like) and a best common denominator.

In operation, the process **1400** begins at block **1410**. At block **1410**, a computing system publishes one or more packages. Each package can include one or more activities. At block **1430**, the type cache service of the computing system scans the feed. The type cache service can also scan all available folders. The type cache service, in implementing a feed management, can proactively execute background scans of one or more available folders and the one or more feeds that are subject to scanning. Further, feed management includes adding and restricting feeds to be scanned. In some cases, the adding and restricting feeds can be in accordance with a type or a group. These one or more scans can be executed periodically, on a time scheduled, and/or triggered.

At block **1440**, the type cache service of the computing system indexes all activities (e.g., new activities and all previously detected activities) according to type. According to one or more embodiments, the type cache service can automatically index/group the one or more unpacked packages according to access. Access grouping can be defined by authorization and/or authentication mechanisms. For example, with respect to enterprise governance, the type cache service can group activities based on department or divisions within the enterprise. The enterprise can include a finance department, a marketing department, a human resources department, a legal department, a sales department, etc. In turn, activities that are meant strictly for the finance department can be restricted from use by other departments (e.g., a certain group should not have access to email activity of legal). By way of example, the computing system can utilize types to exclude a certain group (e.g., non-legal vs. legal departments) from workflows that enable access to exclusive information (e.g., in cases need clearance is required). Further, by way of example, the computing system can utilize types to accommodate system access vs. content access scenarios.

Generally, the process **1400** includes resolving dependencies by building dependency tree and implementing conflict resolutions across the information of the feed, folders, and existing information. At block **1443**, the type cache service of the computing system parses indexed entries for one or more unpacked packages of a type cache. The type cache service builds a dependency tree as the type cache parses indexed entries for the at least new activities. Example of resolving dependencies include, but are not limited to, resolving on user rights, restrict resolving packages based on

the feed, and/or general feed management (e.g., governance, versions, activities, products, etc.)

At block **1445**, the type cache service of the computing system determines a best common denominator of the one or more unpacked packages based on the type cache to utilize in a dependency graph. The common denominator can include when an activity, package, or workflow includes a similar or like factors, traits, or themes, such as pointing to a same preceding activity. At block **1447**, the type cache service of the computing system resolve dependencies of the dependency graph based on the best common denominator.

At block **1450**, the type cache service of the computing system receives one or more inputs. The one or more inputs are utilized to search the indexes (i.e., the type cache). One or more technical, effects, and benefits of the type cache service include enabling the searching of the indexes directly.

At block **1470**, the type cache service of the computing system populates a subset of activities matching the search (i.e., the one or more inputs). At block **1480**, the type cache service of the computing system receives a selection of the populated subset of activities. The selection causes a restoration operation of the corresponding activity.

The computer program can be implemented in hardware, software, or a hybrid implementation. The computer program can be composed of modules that are in operative communication with one another, and which are designed to pass information or instructions to display. The computer program can be configured to operate on a general purpose computer, an ASIC, or any other suitable device.

It will be readily understood that the components of various embodiments, as generally described and illustrated in the figures herein, may be arranged and designed in a wide variety of different configurations. Thus, the detailed description of the embodiments, as represented in the attached figures, is not intended to limit the scope as claimed, but is merely representative of selected embodiments.

The features, structures, or characteristics described throughout this specification may be combined in any suitable manner in one or more embodiments. For example, reference throughout this specification to "certain embodiments," "some embodiments," or similar language means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment. Thus, appearances of the phrases "in certain embodiments," "in some embodiment," "in other embodiments," or similar language throughout this specification do not necessarily all refer to the same group of embodiments and the described features, structures, or characteristics may be combined in any suitable manner in one or more embodiments.

It should be noted that reference throughout this specification to features, advantages, or similar language does not imply that all of the features and advantages that may be realized should be or are in any single embodiment. Rather, language referring to the features and advantages is understood to mean that a specific feature, advantage, or characteristic described in connection with an embodiment is included in one or more embodiments. Thus, discussion of the features and advantages, and similar language, throughout this specification may, but do not necessarily, refer to the same embodiment.

Furthermore, the described features, advantages, and characteristics of the one or more embodiments herein may be combined in any suitable manner. One skilled in the relevant art will recognize that this disclosure can be practiced without one or more of the specific features or advantages of a particular embodiment. In other instances, additional features and advantages may be recognized in certain embodiments that may not be present in all embodiments.

One having ordinary skill in the art will readily understand that this disclosure may be practiced with steps in a different order, and/or with hardware elements in configurations which are different than those which are disclosed. Therefore, although this disclosure has been described based upon these preferred embodiments, it would be apparent to those of skill in the art that certain modifications, variations, and alternative constructions would be apparent, while remaining within the spirit and scope of this disclosure. In order to determine the metes and bounds of this disclosure, therefore, reference should be made to the appended claims.

The invention claimed is:

1. A method executed by a type cache service implemented as a computer program within a computing environment, the method comprising:

scanning, by the type cache service, one or more feeds publishing one or more unpacked packages;

automatically, by the type cache service, detecting at least new activities on the one or more feeds;

indexing, by the type cache service, the at least new activities and all previously detected activities according to type; and

generating, by the type cache service, a type cache for the one or more unpacked packages according to the indexing of the at least new activities and all previously detected activities.

2. The method of claim **1**, wherein the type cache is utilized by the type cache service to provide information on each available activity in the one or more unpacked packages upon a search without a restoration operation.

3. The method of claim **1**, wherein the type cache service proactively executes background scans of one or more available folders and the one or more feeds that are subject to scanning.

4. The method of claim **1**, wherein the scanning by the type cache service includes a plurality of assemblies inside the one or more unpacked packages and metadata attached to the one or more unpacked packages.

5. The method of claim **1**, wherein the type cache service resolves dependencies by building dependency tree for the at least new activities.

6. The method of claim **1**, wherein the type cache is stored in a database accessible by the type cache service.

7. The method of claim **1**, wherein the type cache service receives one or more inputs that are utilized to search the type cache.

8. The method of claim **1**, wherein the type cache service populates a subset of activities matching the one or more inputs and receives a selection of the populated subset of activities to cause a restoration of a corresponding activity.

9. The method of claim **1**, wherein a computing environment publishes the one or more unpacked packages to one or more feeds.

10. The method of claim **1**, wherein the type cache service automatically groups the one or more unpacked packages according to access.

11. A method executed by a type cache service implemented as a computer program within a computing environment, the method comprising:

parsing, by the type cache service, indexed entries for one or more unpacked packages of a type cache;

determining, by the type cache service, a best common denominator of the one or more unpacked packages based on the type cache to utilize in a dependency graph; and

resolving, by the type cache service, dependencies of the dependency graph based on the best common denominator.

12. The method of claim **11**, further comprising:

scanning one or more feeds publishing the one or more unpacked packages;

automatically detecting at least new activities on the one or more feeds; and

indexing, by the type cache service, the at least new activities and all previously detected activities according to type; and

generating, by the type cache service, the type cache for the one or more unpacked packages according to the indexing of the at least new activities and all previously detected activities.

13. The method of claim **11**, wherein the type cache service automatically groups the one or more unpacked packages according to access.

14. The method of claim **11**, wherein the type cache is utilized by the type cache service to provide information on each available activity in the one or more unpacked package upon a search without a restoration operation.

15. The method of claim **11**, wherein the type cache service proactively executes background scans of one or more available folders and the one or more feeds that are subject to scanning.

16. The method of claim **11**, wherein the scanning by the type cache service includes a plurality of assemblies inside the one or more unpacked packages and metadata attached to the one or more unpacked packages.

17. The method of claim **11**, wherein the type cache service resolves dependencies by building dependency tree for the at least new activities.

18. The method of claim **11**, wherein the type cache is stored in a database accessible by the type cache service.

19. The method of claim **11**, wherein the type cache service receives one or more inputs that are utilized to search the type cache.

20. The method of claim **11**, wherein the type cache service populates a subset of activities matching the one or more inputs and receives a selection of the populated subset of activities to cause a restoration of a corresponding activity.

* * * * *