June 2022

CONTEXT-AWARE DYNAMIC CREATION OF LOW-CODE WORKFLOWS FOR TROUBLESHOOTING USE CASES INVOLVING A NETWORK CONTROLLER

Rajesh I V

Divya Puranam

Aditya Kesarwani

Annu Gogayan

Rohit Anand Bhardwaj

CONTEXT-AWARE DYNAMIC CREATION OF LOW-CODE WORKFLOWS FOR TROUBLESHOOTING USE CASES INVOLVING A NETWORK CONTROLLER

AUTHORS:
Rajesh I V
Divya Puranam
Aditya Kesarwani
Annu Gogayan
Rohit Anand Bhardwaj

ABSTRACT

Hyperautomation refers to the combination of available advanced technologies to, as much as possible, strategically automate business processes. The two major technologies that are flourishing in the hyperautomation space are low-code/no-code based applications and robotic process automation (RPA). The networking industry has many automation use cases involving network controllers, especially for troubleshooting, where the technologies that were described above could be widely adopted. However, low-code technologies are currently used for building static applications without any state. Techniques are presented herein that support the dynamic creation of low-code workflows by processing the underlying context and state of networking events for different personas.

DETAILED DESCRIPTION

Year after year, hyperautomation is consistently featured in Gartner's strategic technology trends. In brief, hyperautomation refers to combining all of the available advanced technologies to, as much as possible, strategically automate business processes. The two major technologies that are flourishing in the hyperautomation space are low-code/no-code based applications and robotic process automation (RPA).

Low-code and no-code technologies encompass a visual approach to software development under which users may drag and drop reusable tasks to rapidly construct applications. According to Gartner, more than 65% of application development in 2024 will be performed through low-code platforms.

RPA technologies encompass a mechanism for instructing a machine (e.g., a robot or bot) to execute mundane, repetitive manual tasks. Such software executes the tasks with a predefined set of instructions or rule-based actions.

The networking industry (through, for example, a network controller) has a plethora of automation use cases, especially for troubleshooting, where the technologies that were described above could be widely adopted. Troubleshooting is a major activity where the majority of information technology (IT) administrator time is spent. According to the available data, network operations staff spend more than 65% of their time troubleshooting and fixing issues. The major networking vendors have started leveraging one or both of the above-described technologies for their network management and controller requirements. However, in the current form, the operational philosophies of these technologies impose limitations in achieving all of the network troubleshooting use cases.

A first limitation reflects the reality that low-code applications are primarily used to create static workflows to automate a multitude of independent tasks. There is no inherent design to dynamically create the workflows based on the underlying context and state.

A second limitation reflects the reality that low-code end applications are self-sufficient by design, which means that an application can be hosted independently or work as a plugin in a base product. Troubleshooting workflows will typically have multiple incarnations and personas (such as Developer, Customer Support, and Customer). Examples include workflows for identifying the root cause of issues, reproducing the issues, trying out fixes in a simulated environment, etc. Low-code applications can easily become unmanageable if the network administrator needs to identify the correct task from a huge list of tasks to assemble the different workflows for different personas.

A third limitation reflects the reality that RPA and low-code technologies address various aspects in the automation space. However, there are no available products that combine these technologies together. Network troubleshooting use cases demand a coordination of these two technologies.

In brief, network automation may have a huge list of tasks from which workflows could be created. However, such workflows demand more dynamicity and context for the associated workflows.

To address the type of challenge that was described above, techniques are presented herein that support the dynamic creation of low-code workflows that have the required context and state to triage an underlying problem. Aspects of the presented techniques comprise a number of architectural elements.

A first architectural element encompasses a miniature version of a knowledge graph that captures a low-code tasks knowledge base. Such an artifact, which may be referred to herein as a Tasks Knowledge Graph (TKG), is a graph implementation of nodes and relationships. A TKG may be used in a network controller (within, for example, a digital network architecture environment) to capture detailed information concerning low-code tasks and the possible relationships and trigger patterns with other tasks. Such a navigational pattern provides more insights into how the tasks may be strung together to accomplish a needed troubleshooting.

Under aspects of the techniques presented herein, a TKG captures all of the possible relationships between low-code tasks and events (triggering those tasks) and represents the same as nodes and the relationships between those entities as edges. A TKG may encompass a typical enterprise type knowledge graph which captures the tasks that a network controller exposes. A knowledge graph is highly sophisticated, and many flexible variants exist. Under the techniques presented herein, a TKG may take the form of a directed edge-labelled graph, a heterogeneous graph, a property graph, etc.

It is important to note that a knowledge graph is just one option that may be used to capture tasks and navigational patterns, events, etc. to which an administrator or any artificial intelligence (AI) or machine learning (ML) systems may add more inferences so that with the ontologies any AI, ML, or purpose-built system can glean more intelligence. However, it is important to note that for manageable tasks content and use case scenarios it is possible to use any rich data structures to store the mapping information of tasks and the navigational patterns among tasks.

Figure 1, below, presents a representational diagram for an exemplary TKG.

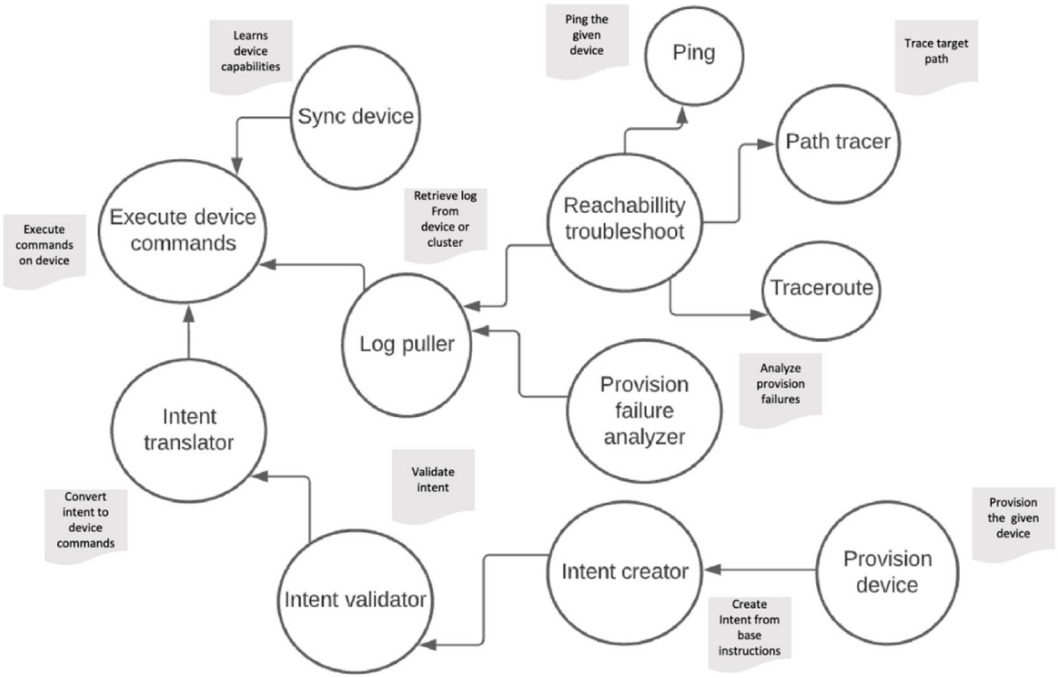

Figure 1: Representational Diagram for an Exemplary TKG

A second architectural element encompasses a Context database (DB) which captures information regarding the runtime context that is associated with the events. The context may include events, underlying network devices, users, locations, time, impact, etc. Aspects of the techniques presented herein support Context Builder pipelines to process the available data such as raw events, log files, etc. to build the context in a Context DB. An administrator may provide the intent of what workflows should be dynamically created as a ruleset. A Context Builder pipeline may also consume such a ruleset and accordingly determine the parsing of the associated data to construct the context.

A third architectural element encompasses a Workflow Compiler Engine (WCE). A WCE component may be used to dynamically build low-code workflows by identifying the correct tasks based on the underlying context and TKG information. Unlike the typical approach of low-code based application creation, a user is not required to know about all of the tasks in order to create the workflow for their specific use case. An administrator may provide minimal information to a WCE (such as which issue needs to be triaged) then the WCE may pull all of the associated context from a Context DB and the tasks correlation from a TKG and then construct a workflow. Alternatively, an administrator may also

provide instructions (which may be stored in a ruleset) if a WCE automatically creates workflows when given conditions are met. Such a WCE task may be handled as an extension of RPA.

Figure 2, below, presents elements of an illustrative architecture according to aspects of the techniques presented herein and reflective of the above discussion.

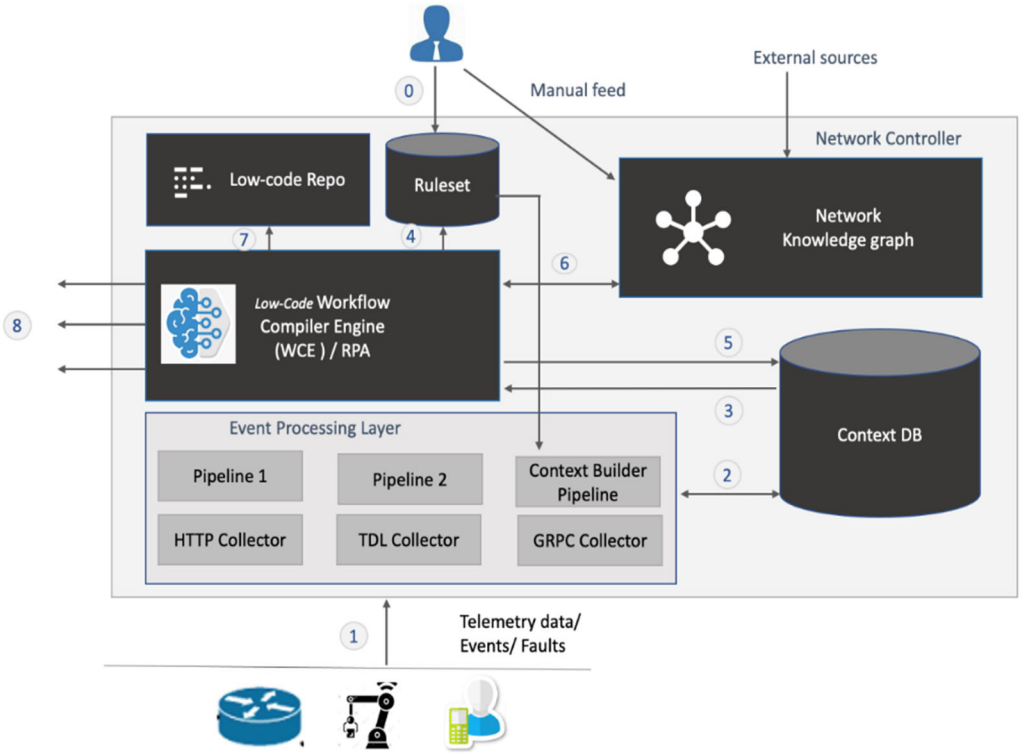

Figure 2: Illustrative Architecture

A WCE is the nerve center of the techniques presented herein and consumes the instructions and intelligence from all of the other subsystems to dynamically compute the low-code workflows. Figure 3, below, depicts various of the functionalities of a WCE.

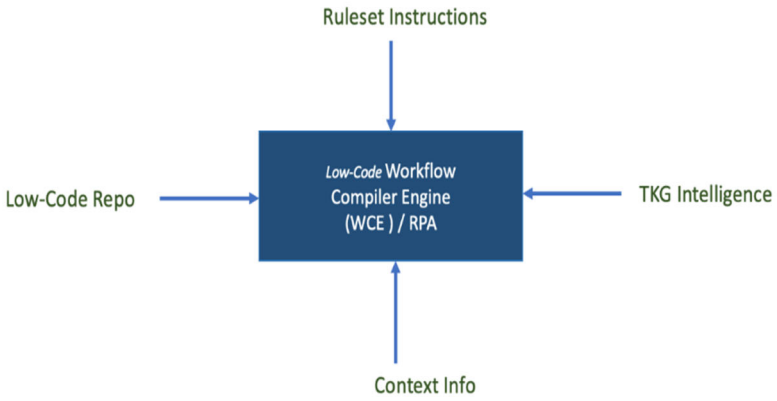

Figure 3: WCE Functionality

A WCE may use the rules that are captured in a ruleset component to identify the kind of problem scenarios for which the troubleshooting workflows are to be dynamically created. This ensures that the dynamic low-code workflows are created only for the given intent of an administrator. A ruleset may be provisioned by an administrator from time to time and also on an on-demand basis. Such a ruleset brings in more control and flexibility to workflow creation. For example, an administrator may establish a rule under which if a slowness is detected in accessing a specific server from a specific region, then only a triaging workflow should be created.

Whenever a WCE identifies a fault scenario that matches the predefined rules it may trigger a workflow creation process. The WCE may gather further insights regarding the underlying fault from the gleaned context that is stored in a Context DB and may determine the combination of tasks that are to be included for dynamically creating a troubleshooting workflow. A TKG contains the intelligence for how the tasks may be linked and the events that are associated to them. The WCE may process that intelligence in determining the workflow. Further, the WCE has access to a low-code repository for the various tasks that are supported for a controller. With the intelligence that is gleaned from the TKG and the context information, the WCE may compile a workflow.

The creation of a workflow (as described above) happens immediately when a WCE identifies a matching condition, with the context that is gleaned from a Context DB

being used for the workflow. Currently, a network controller captures all of the important telemetry (such as faults or key performance indicators (KPIs)) and leverages internal pipelines to process and store the specific context. Since such context is used immediately, the long-term storage of data is not a concern. However, if an administrator wishes to triage an issue that happened some time ago, then there may be a limitation on the availability of data in the database. Typically, a controller may support a history comprising the last six months of data, which supports an acceptable troubleshooting timespan.

As described and illustrated in the above narrative, the techniques presented herein support the dynamic creation of low-code applications with associated state context. Such an arrangement is a primary requirement for troubleshooting. As an analogy, aspects of the presented techniques position a WCE more like an RPA module with additional intelligence (that is captured as described above).

The techniques presented herein may be further understood through an exemplary workflow. The steps of that workflow will be described below in connection with the illustrative architecture that was depicted in Figure 2, above, and the illustrative sequence diagram that is presented in Figure 4, below.

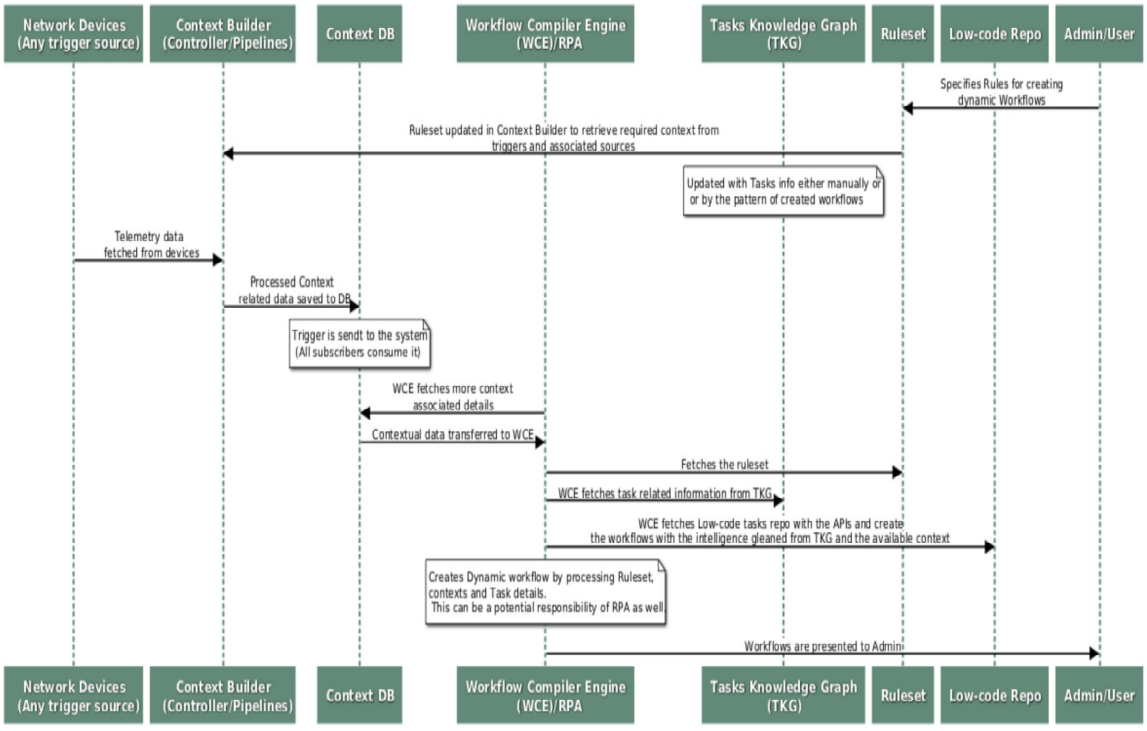

Figure 4: Illustrative Sequence Diagram

During Step 0, two options may be available for instructing a controller to dynamically create a workflow. Under a first option, an administrator may provide instructions upfront to the system regarding the matching conditions for which the troubleshooting workflows are to be created. Those instructions may contain the event triggers that were detected, internal state changes or problems that were identified by application performance management (APM) facilities, etc. Such instructions may be captured as a ruleset. Under a second option, an administrator may log on to the system and provide instructions to create a workflow to troubleshoot a problem that has previously occurred. The administrator may provide more information on the low-code tasks and the associated intelligence to a TKG as a manual feed.

During Step 1, a controller may be subscribed to receive telemetry (such as events, faults, KPIs, etc.) from network devices. The trigger of sources may be multiple in a typical network controller.

During Step 2, an event processing layer (such as data collectors and transforming pipelines) may process the data, drill down further, and collect additional context associated with the data and store same in a Context DB. A Context Builder pipeline may consume the ruleset that was instructed by an administrator and ensure that the required runtime context and state are captured for the event triggers.

For simplicity of exposition, the illustrative architecture that was presented in Figure 2, above, depicted device telemetry. However, it is important to note that an event source may be anything (e.g., south bound, north bound, or east-west interactions) and that configuration provisioning errors, the internal fault events of networking issues, etc. may all be counted during Step 2.

During Step 3, a Context DB may fire an event off to the system regarding a Context DB update so that all of the subscribers may consume and act upon it. Under Step 4, a WCE – as a subscriber to the event – may check the context event and verify with the rule instructions if any workflows need to be dynamically created for triaging the issue. Then during Step 5, the WCE may fetch additional associated details for the context.

During Step 6, the WCE may query a TKG and collect all of the knowledge information regarding the triggers and the associated low-code tasks. Under Step 7, the

WCE may fetch from the low-code tasks repository (e.g., through application programming interfaces (APIs)) and create the workflows with the intelligence that was gleaned from the TKG and the available context. Note that further examples will be described below in connection with a discussion of the different use cases to which aspects of the techniques presented herein may be applied.

Finally, during Step 8 the workflows are dynamically created with the associated context and then presented to the user.

As described an illustrated in the above discussion of an exemplary workflow, the techniques presented herein focus primarily on supporting the dynamic generation of low-code workflows comprising underlying context and state. Aspects of the presented techniques encompass RPA working as a WCE and bridge the technologies of RPA and low-code/no-code.

The techniques presented herein may be further explicated through an illustrative end-to-end example comprising an enterprise user who is experiencing slowness or packet losses while accessing a business critical application. That example will be described in the next portion of the instant narrative.

Before presenting a detailed discussion of the illustrative end-to-end example, it will be helpful to briefly reiterate some of the challenges that arise with low-code applications and other existing comparable solutions.

Low-code platforms are primarily focused on the rapid construction of applications. Such a focus promotes citizen development, where applications are statically created by dragging and dropping the required tasks. Such a low-code approach is not focused on dynamically building applications through the stitching together of the required tasks. Typically, dynamic applications require that state information and runtime context be passed into the applications, which (as noted previously) is not what low-code platforms are primarily meant for. However, the low-code concept represents a powerful approach for creating applications which, according to the techniques presented herein, may be extended to network troubleshooting use cases (and which address all of the above-described limitations). For network troubleshooting, there may exist a plethora of individual task units from which applications may be constructed and, moreover, such troubleshooting efforts require the actual states and context.

As noted previously, RPA and low-code technologies address various aspects in the automation space. However, there are no available products that combine these technologies together. And importantly, network troubleshooting use cases demand a coordination of these two technologies.

To provide important context, the instant end-to-end example (comprising an enterprise user who is experiencing slowness or packet losses while accessing a business critical application) will first be discussed from the point of view of how the instant issue might be addressed using an existing low-code platform

Ideally, the approaches that are followed by a user to troubleshoot the instant issue include, for example, checking the target server reachability, checking the path to reach the server, checking the application policy specific configuration, checking the Dynamic Host Configuration Protocol (DHCP) marking of the traffic at various points, checking device health and congestion patterns, analyzing the logs of the devices, etc.

If a low-code platform is used to troubleshoot the issue, then the platform needs to expose a range of tasks (similar to the user approaches that were described above) for building troubleshooting applications which a network administrator may then drag and drop to construct tailor-made troubleshooting applications. For example, the tasks that may be needed for the instant example include (where the exemplary tasks names are self-explanatory) ping-target-device/application, path-tracer, debug-log-analyzer, dhcp-marking-checker, device-command-executor, device-health-monitor, etc.

Under the approach that was described above, an administrator faces a number of challenges. First, the administrator needs to be fully aware of the different low-code tasks, the usage and trigger patterns, and how to pass inferences from one to another. Second, the administrator needs to obtain the required context and state information to pass same onto an application. Here, the context could be which user is facing the issue, which target application is involved, from which location the problem arises, etc. Third, triaging the error is not accomplished in a timely manner. Consequently, certain real-time context is missed at the time of troubleshooting. As a result, the above-described approach is not efficient.

Next, the instant end-to-end example will be discussed from the point of view of how aspects of the techniques presented herein may be employed to address the slowness or packet loss issue.

The above discussion of how the instant issue might be addressed using an existing low-code platform clearly identified a number of missing elements which make that approach less convenient for troubleshooting use cases.

A first missing element is an inbuilt knowledge base of the low-code tasks (i.e., information regarding the relationships between tasks, trigger sources of tasks etc.). A second missing element is a mechanism for capturing the required context and states which are relevant for troubleshooting. A third missing element is the identification of the trigger point for determining the issue which needs troubleshooting. A system requires a policy with a set of instructions to corelate and determine the interesting error conditions to avoid false positives.

The above-described elements can be handled using novel techniques, as described herein. First, a troubleshooting intent (that is captured as a ruleset as described and illustrated above) may be provided to a low-code capable network controller that dictates matching issue scenarios for which troubleshooting applications are to be automatically created. Such intent may be dynamic in nature with associated instructions.

For the instant example, an administrator may specify an intent such as "Business critical application access latency is to be triaged." The intent will have associated instructions such as accepted latency threshold, etc.

Second, a Context Builder layer processes the given intents and converts them into actionable tasks in the controller. Actionable tasks may include monitoring the intent-specific scenarios, capturing the required contexts, and generating trigger events for computing a troubleshooting application if the issue scenario is detected.

For the instant example, once the intent is processed the internal system may be programmed to monitor all business-critical application access against the given latency threshold. The Context Builder pipeline may begin capturing the states that are specific to this intent (such as which client Internet Protocol (IP) address is accessing the application, from which location, etc.) with all of the relevant details updated in a data store. When the monitoring system detects a problem scenario (here, the access of a business critical

application from a particular location from a particular client falls below an expected threshold mark) a fault trigger event is generated to triage the specific problem.

Third, a WCE, as a consumer of the trigger event, begins compiling a troubleshooting workflow. Low-code knowledge management is a crucial aspect of the automated creation of a troubleshooting application, with well-organized data easing any analysis steps. In connection with that need, aspects of the techniques presented herein support the capture of the knowledge regarding low-code tasks in a TKG as ontologies with definitions, relationships, triggers, rules, etc. Importantly, a TKG may be computed with manual feeds, vendor feeds through a cloud, ML programs processing the data usage patterns, etc.

As first step of analysis, a WCE may determine if the underlying error is centered around a single device or if it is network wide. Accordingly, an error topology may be created. For example, if the error type is related to a network configuration, then it may pertain to a device. In contrast, for the instant example the error could happen anywhere along a path. The topology captures a list of the devices that need to be checked. Low-code tasks may be determined and then an application may be created to check on those devices.

For the instant example, a WCE may first consume intelligence from a TKG. Under the techniques presented herein, low-code tasks, devices, applications, etc. may be captured as nodes and a relationship may capture how one low-code node is linked with another and what events require a specific task. Such TKG information may be encapsulated in a table, an example of which is presented in Table 1, below.

Table 1: Exemplary TKG Information

Node	Relationship	Node
Device	Reachability	ping_device (low-code task)
Device	Path to reach	path_trace (low-code task)
dhcp_checker (low-code task)	Uses	packet_capture (low-code task)
log_capture	Uses	log_analyzr (low-code task)
Device	Executes	command_executor (low-code task)

A WCE may identify what will be the first task in a troubleshooting workflow and, accordingly, it may compute the follow up tasks and build a chain with conditional checks being inserted as needed. A TKG may be further used to chase down the balance of the tasks in compiling the workflow. The WCE may programmatically create the application by invoking the respective APIs of the tasks. Further, the WCE may also obtain the required context from a Context DB, apply that information to the application as required, and trigger the application for troubleshooting.

For the instant example, a sequence as described above might begin with:

ping-target-device/application ->
path-tracer ->
device-command-executor ->
debug-loganalyzer ->
dhcp-marking-checker ->

and then continue on.

It is important to note that the exemplary workflow and the illustrative end-to-end example that were presented above are for purposes of exposition. Every troubleshooting scenario will be unique by its nature. A WCE may make use of existing graph analytics approaches to query and process the data. However, the techniques presented herein demonstrate how an RPA system can work with a low-code system with a TKG serving as a bridge.

As described and illustrated in the above narrative, the techniques presented herein may provide a number of features. For example, techniques herein may facilitate the dynamic creation of context-aware, tailor-made, low-code applications with gleaned states for a given troubleshooting intent. Further, techniques herein may facilitate the construction of intent-based, low-code applications within a controller for troubleshooting use cases. Additionally, techniques herein may enable the immediate triage of a network issue through the dynamic triggering of a purpose-built application with the captured context. Moreover, techniques herein may provide for the exposure of low-code tasks as a TKG (which captures the relationships among tasks, trigger patterns, etc.) as a graph to aid any graph analytics facility in utilizing the data to glean further insights.

The techniques presented herein may be applied to a number of different use cases, including an investigation into network provisioning failures, an analysis of network slowness when a speed drops below a threshold, an investigation into critical alerts from a device and the identification of a cause and impact, etc.

According to aspects of the presented techniques, a network administrator may specify the type of issues, events, etc. for which automatic troubleshooting workflows are to be created when an issue is detected. Such workflows may be created for different purposes, for instance troubleshooting an issue, reproducing the issue, applying a fix and verifying the issue, etc. Additionally, there may be different actors such as Developer, Customer Support, and Customer. Further, a network administrator may dynamically adjust a ruleset to enable or disable workflow creation.

In connection with the creation of a workflow to troubleshoot an issue (as described above), if a rule exists regarding the creation of a workflow to triage an underlying problem, then the workflow may be created with the context when the issue is triggered.

For example, a rule may be defined for the triage of a user connection error. If a failure is detected while a user connecting to the network, an automatic troubleshooting workflow may be created to check the status from an access point (AP), an authentication in an identity services engine, etc. A WCE may automatically connect all of the tasks that are required for triaging such an error and then pass the context. A ‘user id’ may encompass the context or state information.

Alternatively, if there is no such rule specified, an administrator may log in and ask to trigger the authentication failure of the user by providing a ‘user id’ after which the workflow may be dynamically created to triage the error.

Under the above example, the different tasks may include an AP attachment check, an identity services engine authentication check, a policy check, a log tracer, etc.

In connection with the creation of a workflow to reproduce an issue (as described above), if a customer faces an issue and wishes to be triaged by a network vendor, then on the customer’s side the customer may establish a rule, reproduce the issue, and the background workflow may be created to simulate the issue with the required context. The customer may then share the workflow with the network vendor so that the vendor may execute and reproduce the issue.

As described and illustrated in the above narrative, the techniques presented herein encompass a number of capabilities. A first capability comprises the dynamic creation of low-code workflows through learned internal triggers and context and state. A second capability comprises the ability to represent, and continuously update, the low-code tasks and the associated trigger and relationship information in a TKG. A third capability comprises the ability to specify dynamic instructions to a system to create low-code workflows based on a given ruleset. A fourth capability comprises combining RPA and low-code technologies (so that those technologies may work together) where a WCE component may be offered through RPA.

As noted previously, the networking industry raises a significant number of use cases for hyperautomation, especially in the troubleshooting space, through the power of low-code applications and RPA. The dynamic construction of workflows for troubleshooting use cases, according to the techniques presented herein, will be of great value to network equipment vendors through improvements to the customer experience and reductions in turnaround time.

In summary, techniques have been presented that support the dynamic creation of low-code workflows by processing the underlying context and state of networking events for different personas. Aspects of the presented techniques encompass a TKG to capture detailed information concerning low-code tasks and the possible relationships and trigger patterns with other tasks; a Context DB which captures information regarding a runtime context (which may include events, underlying network devices, users, locations, time, impact, etc.) that is associated with events; Context Builder pipelines to process the available data (such as raw events, log files, etc.) to build the context in a Context DB; and a WCE which may be used to dynamically build low-code workflows by leveraging the associated context in a Context DB and the tasks correlation in a TKG.