Directed Acyclic Graphs and its application on Workflow Management

Guillermo Wrba
Autor de "Designing and Building Solid Microservice Ecosystems", Consultor...
13 articles + Follow

December 29, 2021

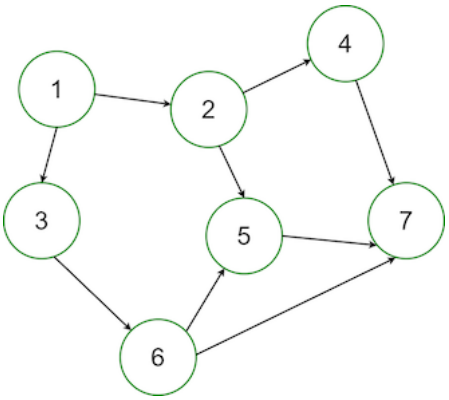
Directed Acyclic Graph is not a new concept. In fact, the concept of DAGs and their applications have been widely analyzed and explored by John Pfaltz and then by other mathematicians, especially in fields related to geometry, spatial analysis, walk/path analysis and is intrinsically part of the universal graph theory.

As per Wikipedia:

> In mathematics, particularly graph theory, and computer science, a directed acyclic graph is a directed graph with no directed cycles. That is, it consists of vertices and edges (also called arcs), with each edge directed from one vertex to another, such that following those directions will never form a closed loop. A directed graph is a DAG if and only if it can be topologically ordered, by arranging the vertices as a

> linear ordering that is consistent with all edge directions. DAGs have numerous scientific and computational applications, ranging from biology (evolution, family trees, epidemiology) to sociology (citation networks) to computation (scheduling).

In summary, a DAG represents a set of nodes and their relationships, as we can see on the below image:



From this sample graph, we can clearly see why we call "acyclic" and it's because there is no way to come back any of the nodes, starting from any position (1 to 7). in other words, there are no chances to get into a "loop" when traversing the graph. This is the basic condition for a DAG.

There are an inifinite number of applications of directed-acyclic-graphs and throughout this article, i'll be exploring how we can leverage its concepts in order to implement a phased-out, DAG-oriented, backward/forward chained workflow. These terms may sound complicate, but in fact, they are not. Let me explain what i'm referring to....

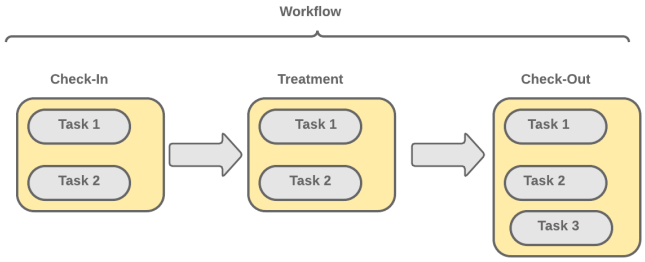
The concept of a phased-out worfklow

In terms of compute science, a workflow represents a set of activities that are chained together in a particular sequence, that necessarily a certain actor must pass through in a sequenced mode in order to accomplish it. By following this very open definition, there could be multiple different types of workflows, depending on the particular field we are studying. For example, there could be human

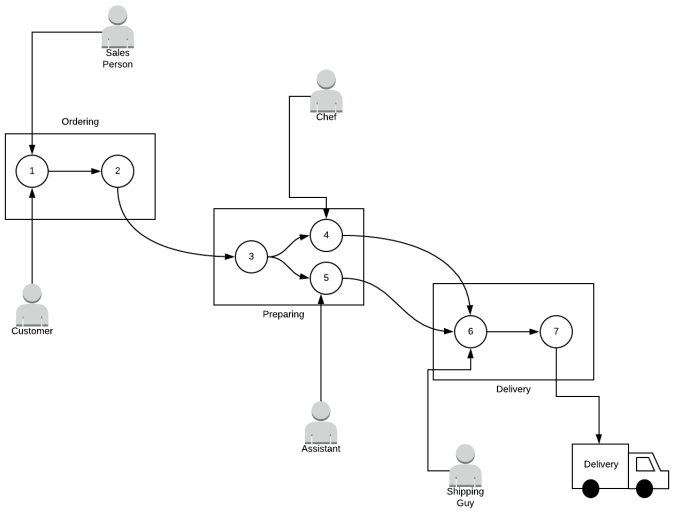
workflows, manufacturing workflows, assembly workflows, data processing workflows, and each of them can have their own nature because they belong to different problem spaces.

A phased-out workflow represents a whole workflow that has been divided into two or more sub-workflows which means the whole set of activities gets divided up; actors must complete the first sub-workflow as a prerequisite to complete the second sub-workflow, and that applies to all the workflow chain. Each sub-workflow is considered a "phase".

The below diagram depicts a doctor visit workflow, for which there have been identified three different phases, one for check-in for patient identification and data collection , another for treatment or visit, and a check-out phase for money collect and prescription dispense. Each phases has their own tasks, as the diagram depicts, so the tasks for check-in must be completed before the patient can go through the treatment phase:



The above is a very simplified model of a real workflow model, of course; one important thing we forgot to mention is that not all of the tasks (or activities) belonging to a certain phase are mandatory for phase completion; in fact, in a real-world scenario only a sub-set of the tasks can be considered as "mandatory" for phase progression.



Workflow and sub-workflow State

When we talk about "phase progression" we are also intrinsecally thinking about some workflow state associated with the phase, and with the overall workflow; workflow phases can be in three different states: not started, in progress and complete:

- Not started: any mandatory activity (milestone) required to start the current phase hasn't been completed yet. A phase starts only when all the prerequisite tasks have been completed.
- in progress: All mandatory required tasks for the current phase to start have been accomplished, however yet not all the tasks belonging to the current phase have been accomplished.
- complete: all the tasks (activities) related to the current phase have been accomplished.

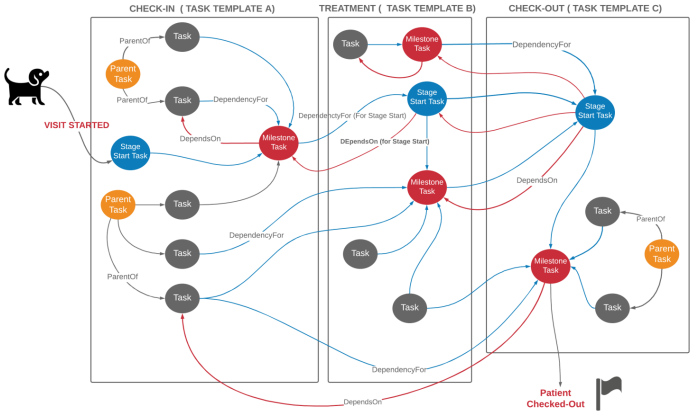
We above highlighted the fact that not all the tasks are mandatory for phase progression in a workflow; such tasks that are mandatory are known as "milestone" tasks or activities. A milestone can be a dependency for another phase or phases to transition from the "not started" into the "in progress" state.

Task Types in a DAG-based workflow model

The below diagram depicts the different types of nodes we c find on a typical phased-out workflow model:

- Simple Task Nodes: simple tasks represents activities that belong to a certain workflow phase.
- Parent Task Nodes: tasks can be represented through a parent-child hierarchy whenever parent and child are related together. i.e. in the check-in example we can have a parent task for "verify patient information" which can have multiple children for "verify patient weight", "verify patient age" and "verify patient medical history". Both are interrelated, but the parent encompasses all the child's activities. A parent node is considered completed only if all the children activities are in a completed state.
- Child Task Nodes: a parent task is such that can have one or more related child nodes. If a certain child node is not completed, the parent must automatically flagged as not completed.
- Start Node: a start node represents an automatic task that once completed, proceeds to transition the phase to which it belongs, from "not started" into the "in progress" state. Since a start node depends on one or more child tasks from different phases, it can only transition into the completed state only if all of its dependant tasks are flagged as completed. A phase formally starts once the start node is flagged as completed. In an opposite direction, if any of its dependant tasks are flagged as not completed, then the start node is flagged as not completed as well, and the linked phase must be transitioned from the "in progress" into the "not started" state.
- Milestone Node: a milestone necessarily represents a forward dependency for another phase's start node, and itself may have multiple backward dependencies to one or more child or simple tasks. Only once all the dependencies have been completed, the milestone can become completed, and hence, the next phase's start node could be also flagged as completed, triggering the next phase to become in progress (from not started). A milestone represents a crucial

concept in this phased-out dag workflow, since it is where the real workflow becomes triggered and phase progression becomes materialized.



In the above diagram, a whole patient visit workflow is depicted, which gets divided into three main sub-workflows. Each sub-workflow can have different parent and child activities. Such activities can become a dependency for a milestone which itself becomes a dependency for another phase's start node. The chaining between parent tasks, tasks, milestone and start nodes is what drives the workflow progression and is known as **forward and backward chaining.**

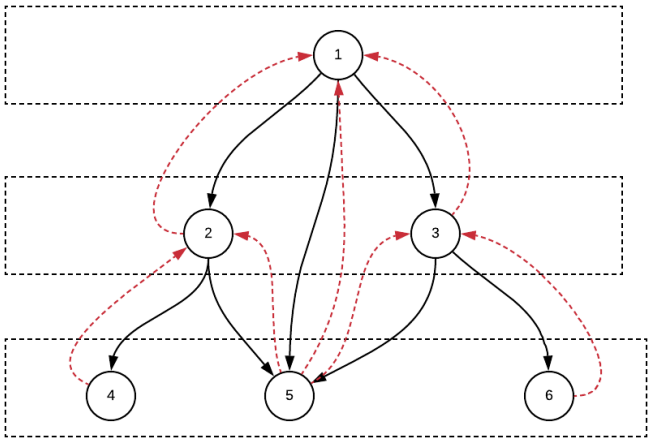
Forward and backward chaining

Forward and backward chaining is the "glue" that keeps the different nodes connected together by means of different kind of relationships and helps to minimize the burden associated with the phased-out workflow computation. In a typical DAG, phased workflow we can find the below relationships:

- **ParentOf** relationship: applies only to parent node. This relationship enumerates all the corresponding node children together. If we name A as the parent and B as the child set, it's an A->B relationship.
- **DependencyFor** relationship: this represents the "forward chaining" and applies to both simple and child nodes. This relationship enumerates all the corresponding milestone nodes for which the node

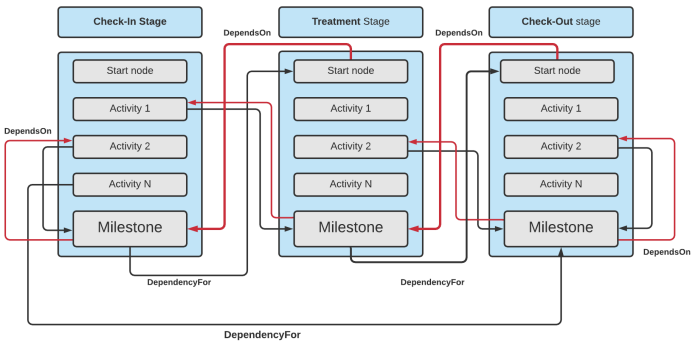
becomes a workflow dependency. If we name A as the node and B as the milestone set, it's an A->B relationship.

- **DependsOn** relationship: it's the opposite to the above one, and represents the "backward chaining", that can be applied mainly to milestones. This relationship enumerates all the associated nodes on top of which this milestone depends on for completion. If we name A as the milestone and B as the node set, it's an A->B relationship.



Summarizing: the high-level view

The below sample diagram summarizes all the concepts we have discussed above into a single high-level view of the workflow, including phases , tasks (activities), milestones (stage progression) and the forward-backward chaining relationships among them.



Wrapping up

There are numerous real-world applications for the DAG phased-out approach that has been presented in this article, being one of the most important the checklist-based workflow, for which users are directed through a certain set of phased-out checklists whose workflow is primarily driven by the options being checked / unchecked. This use case can typically cover the checklists used by doctors in a typical hospital in order to accomplish with their daily duties, for which a patient must be tracked across multiple stages.

In the next article i'll explain how can we implement a data model that could support this approach, and how we can certainly implement a succesfull workflow engine that would support the execution of the workflow rules by means of forward/backward chaining..

Coming soon !!...

Moving Mission-Critical SQL Server Workloads Effectively to Cloud with...

Guillermo Wrba on LinkedIn

Architecture Deliverables Governance Framework [AGDF]

Guillermo Wrba on LinkedIn

Published on LinkedIn

ML/AI, un hijo de modelos de regresion matematica

Guillermo Wrba on LinkedIn

See all 13 articles