Many cloud applications require the completion of multiple interdependent tasks; the description of a

complex activity involving such an ensemble of tasks is known as a workflow. In this section we discuss

workflow models, the life cycle of a workflow, the desirable properties of a workflow description,

workflow patterns, reachability of the goal state of a workflow, and dynamic workflows and conclude

with a parallel between traditional transaction systems and cloud workflows [230].

Workflow models are abstractions revealing the most important properties of the entities participating

in a workflow management system. Task is the central concept in workflow modeling; a task is a unit

of work to be performed on the cloud, and it is characterized by several attributes, such as:

• Name. A string of characters uniquely identifying the task.

• Description. A natural language description of the task.

• Actions. Modifications of the environment caused by the execution of the task.

• Preconditions. Boolean expressions that must be true before the action(s) of the task can take place.

• Post-conditions. Boolean expressions that must be true after the action(s) of the task take place.

• Attributes. Provide indications of the type and quantity of resources necessary for the execution of

the task, the actors in charge of the tasks, the security requirements, whether the task is reversible,

and other task characteristics.

• Exceptions. Provide information on how to handle abnormal events. The exceptions supported by a

task consist of a list of <event, action> pairs. The exceptions included in the task exception

list are called anticipated exceptions, as opposed to unanticipated exceptions. Events not included

in the exception list trigger replanning. Replanning means restructuring of a process or redefinition

of the relationship among various tasks.

A composite task is a structure describing a subset of tasks and the order of their execution. A primitive

task is one that cannot be decomposed into simpler tasks. A composite task inherits some properties

from workflows; it consists of tasks and has one start symbol and possibly several end symbols. At

the same time, a composite task inherits some properties from tasks; it has a name, preconditions, and

post-conditions.

A routing task is a special-purpose task connecting two tasks in a workflow description. The task

that has just completed execution is called the predecessor task; the one to be initiated next is called

the successor task. A routing task could trigger a sequential, concurrent, or iterative execution. Several

types of routing task exist:

• A fork routing task triggers execution of several successor tasks. Several semantics for this construct

are possible:

• All successor tasks are enabled.

• Each successor task is associated with a condition. The conditions for all tasks are evaluated,

and only the tasks with a true condition are enabled.

• Each successor task is associated with a condition. The conditions for all tasks are evaluated,

but the conditions are mutually exclusive and only one condition may be true. Thus, only one

task is enabled.

• Nondeterministic, k out of n > k successors are selected at random to be enabled.

• A join routing task waits for completion of its predecessor tasks. There are several semantics for the

join routing task:

• The successor is enabled after all predecessors end.

• The successor is enabled after k out of n > k predecessors end.

• Iterative: The tasks between the fork and the join are executed repeatedly.

A process description, also called a workflow schema, is a structure describing the tasks or activities

to be executed and the order of their execution. A process description contains one start symbol and

one end symbol. A process description can be provided in a workflow definition language (WFDL),

supporting constructs for choice, concurrent execution, the classical fork, join constructs, and iterative

execution. Clearly, a workflow description resembles a flowchart, a concept we are familiar with from

programming.

The phases in the life cycle of a workflow are creation, definition, verification, and enactment. There

is a striking similarity between the life cycle of a workflow and that of a traditional computer program,

namely, creation, compilation, and execution (see Figure 4.1). The workflow specification by means of

a workflow description language is analogous to writing a program. Planning is equivalent to automatic

program generation. Workflow verification corresponds to syntactic verification of a program, and

workflow enactment mirrors the execution of a compiled program.

A case is an instance of a process description. The start and stop symbols in the workflow description

enable the creation and the termination of a case, respectively. An enactment model describes the steps taken to process a case. When a computer executes all tasks required by a workflow the enactment can

be performed by a program called an enactment engine.

The state of a case at time t is defined in terms of tasks already completed at that time. Events cause

transitions between states. Identifying the states of a case consisting of concurrent activities is considerably more

difficult than identifying the states of a strictly sequential process. Indeed, when several activities

could proceed concurrently, the state has to reflect the progress made on each independent activity.

An alternative description of a workflowcan be provided by a transition system describing the possible

paths from the current state to a goal state. Sometimes, instead of providing a process description, we

may specify only the goal state and expect the system to generate a workflow description that could

lead to that state through a set of actions. In this case, the new workflow description is generated

automatically, knowing a set of tasks and the preconditions and post-conditions for each one of them.

In artificial intelligence (AI) this activity is known as planning.

The state space of a process includes one initial state and one goal state; a transition system identifies

all possible paths from the initial to the goal state. A case corresponds to a particular path in the transition

system. The state of a case tracks the progress made during the enactment of that case.

Among the most desirable properties of a process description are the safety and liveness of the

process. Informally, safety means that nothing “bad” ever happens, and liveness means that something

“good” will eventually take place should a case based on the process be enacted. Not all processes are

safe and live. For example, the process description in Figure 4.2(a) violates the liveness requirement.

As long as task C is chosen after completion of B, the process will terminate. However, if D is chosen,

then F will never be instantiated, because it requires the completion of both C and E. The process will

never terminate, because G requires completion of both D and F.

A process description language should be unambiguous and should allow a verification of the process

description before the enactment of a case. It is entirely possible that a process description may be enacted

correctly in some cases but fail for others. Such enactment failures may be very costly and should be

prevented by a thorough verification at the process definition time. To avoid enactment errors, we need to

verify process description and check for desirable properties such as safety and liveness. Some process

description methods are more suitable for verification than others.

A note of caution: Although the original description of a process could be live, the actual enactment

of a case may be affected by deadlocks due to resource allocation. To illustrate this situation, consider

two tasks, A and B, running concurrently. Each of them needs exclusive access to resources r and q for

a period of time. Either of two scenarios is possible:

1. A or B acquires both resources and then releases them and allows the other task to do the same.

2. We face the undesirable situation in Figure 4.2(b) when, at time t1, task A acquires r and continues

its execution; then at time t2 task B acquires q and continues to run. Then at time t3 task B attempts

to acquire r and it blocks because r is under the control of A. Task A continues to run and at time

t4 attempts to acquire q and it blocks because q is under the control of B.

The deadlock illustrated in Figure 4.2(b) can be avoided by requesting each task to acquire all

resources at the same time. The price to pay is underutilization of resources. Indeed, the idle time of

each resource increases under this scheme.

Workflow pattern refers to the temporal relationship among the tasks of a process. The workflow

description languages and the mechanisms to control the enactment of a case must have provisions to support these temporal relationships. Workflow patterns are analyzed in [1,382]. These patterns

are classified in several categories: basic, advanced branching and synchronization, structural, statebased,

cancellation, and patterns involving multiple instances. The basic workflow patterns illustrated

in Figure 4.3 are:

• The sequence pattern occurs when several tasks have to be scheduled one after the completion of

the other [see Figure 4.3(a)].

• TheAND split pattern requires several tasks to be executed concurrently. Both tasks B and C are

activated when task A terminates [see Figure 4.3(b)]. In case of an explicit AND split, the activity

graph has a routing node and all activities connected to the routing node are activated as soon as the

flow of control reaches the routing node. In the case of an implicit AND split, activities are connected

directly and conditions can be associated with branches linking an activity with the next ones. Only

when the conditions associated with a branch are true are the tasks activated.

• The synchronization pattern requires several concurrent activities to terminate before an activity can

start. In our example, task C can only start after both tasks A and B terminate [see Figure 4.3(c)].

• TheXOR split requires a decision; after the completion of task A, either B or C can be activated

[see Figure 4.3(d)].

• In theXOR join, several alternatives are merged into one. In our example, task C is enabled when

either A or B terminates [see Figure 4.3(e)].

• TheOR split pattern is a construct to choose multiple alternatives out of a set. In our example, after

completion of task A, one could activate either B or C, or both [see Figure 4.3(f)].

• The multiple merge construct allows multiple activations of a task and does not require synchronization

after the execution of concurrent tasks. Once A terminates, tasks B and C execute concurrently [see Figure 4.3(g)]. When the first of them, say, B, terminates, task D is activated; then when C

terminates, D is activated again.

• The discriminator pattern waits for a number of incoming branches to complete before activating

the subsequent activity [see Figure 4.3(h)]; then it waits for the remaining branches to finish without

taking any action until all of them have terminated. Next, it resets itself.

• TheN out of M join construct provides a barrier synchronization. Assuming that M > N tasks run

concurrently, N of them have to reach the barrier before the next task is enabled. In our example,

any two out of the three tasks A, B, and C have to finish before E is enabled [see Figure 4.3(i)].

• The deferred choice pattern is similar to the XOR split, but this time the choice is not made explicitly

and the run-time environment decides what branch to take [see Figure 4.3(j)].

Next we discuss the reachability of the goal state and we consider the following elements:

• A system  , an initial state of the system, σini tial , and a goal state, σgoal .

• A process group P = {p1, p2, . . . , pn}; each process pi in the process group is characterized by a

set of preconditions, pre(pi ), post-conditions, post(pi ), and attributes, atr(pi ).

• A workflow described by a directed activity graph A or by a procedure   capable of constructing

A given the tuple < P, σini tial, σgoal >. The nodes of A are processes in P and the edges define

precedence relations among processes. Pi → Pj implies that pre(p j ) ⊂ post(pi ).

• A set of constraints C = {C1,C2, . . . ,Cm}.

The coordination problem for system   in state σini tial is to reach state σgoal as a result of postconditions

of some process Pf inal ∈ P subject to constraints Ci ∈ C. Here σini tial enables the preconditions

of some process Pini tial ∈ P. Informally, this means that a chain of processes exists such that

the post-conditions of one process are preconditions of the next process in the chain.

Generally, the preconditions of a process are either the conditions and/or the events that trigger the

execution of the process or the data the process expects as input; the post-conditions are the results

produced by the process. The attributes of a process describe special requirements or properties of the

process.

Some workflows are static. The activity graph does not change during the enactment of a case.

Dynamic workflows are those that allow the activity graph to be modified during the enactment of

a case. Some of the more difficult questions encountered in dynamic workflow management refer to

(i) how to integrate workflow and resource management and guarantee optimality or near optimality

of cost functions for individual cases; (ii) how to guarantee consistency after a change in a workflow;

and (iii) how to create a dynamic workflow. Static workflows can be described in WFDL (the workflow

definition language), but dynamic workflows need a more flexible approach.

We distinguish two basic models for the mechanics of workflow enactment:

1. Strong coordination models, whereby the process group P executes under the supervision of a

coordinator process or processes. A coordinator process acts as an enactment engine and ensures a

seamless transition from one process to another in the activity graph.

2. Weak coordination models, whereby there is no supervisory process.

In the first case, we may deploy a hierarchical coordination scheme with several levels of coordinators.

A supervisor at level i in a hierarchical scheme with i +1 levels coordinates a subset of processes

in the process group. A supervisor at level i − 1 coordinates a number of supervisors at level i and the

root provides global coordination. Such a hierarchical coordination scheme may be used to reduce the

communication overhead; a coordinator and the processes it supervises may be colocated.

The most important feature of this coordination model is the ability to support dynamic workflows.

The coordinator or the global coordinator may respond to a request to modify the workflow by first

stopping all the threads of control in a consistent state, then investigating the feasibility of the requested

changes, and finally, implementing feasible changes.

Weak coordination models are based on peer-to-peer communication between processes in the process

group by means of a societal service such as a tuple space. Once a process pi ∈ P finishes, it

deposits a token, including possibly a subset of its post-conditions, post(pi ), in a tuple space. The

consumer process p j is expected to visit the tuple space at some point in time, examine the tokens

left by its ancestors in the activity graph, and, if its preconditions pre(p j ) are satisfied, commence the

execution. This approach requires individual processes to either have a copy of the activity graph or

some timetable to visit the tuple space. An alternative approach is using an active space, a tuple space

augmented with the ability to generate an event awakening the consumer of a token.

There are similarities and some differences between workflows of traditional transaction-oriented

systems and cloud workflows. The similarities are mostly at the modeling level, whereas the differences

affect the mechanisms used to implement workflow management systems. Some of the more subtle

differences between the two are:

• The emphasis in a transactional model is placed on the contractual aspect of a transaction; in a

workflow the enactment of a case is sometimes based on a “best-effort” model whereby the agents

involved will do their best to attain the goal state but there is no guarantee of success.

• A critical aspect of the transactional model in database applications is maintaining a consistent state

of the database; however, a cloud is an open system, and thus its state is considerably more difficult

to define.

• The database transactions are typically short-lived; the tasks of a cloud workflow could be long lasting.

• A database transaction consists of a set of well-defined actions that are unlikely to be altered during

the execution of the transaction. However, the process description of a cloud workflow may change

during the lifetime of a case.

• The individual tasks of a cloud workflow may not exhibit the traditional properties of database

transactions. For example, consider durability: At any instance of time, before reaching the goal

state, a workflow may roll back to some previously encountered state and continue from there on an

entirely different path. A task of a workflow could be either reversible or irreversible. Sometimes,

paying a penalty for reversing an action is more profitable in the long run than continuing on a wrong

path.

• Resource allocation is a critical aspect of the workflow enactment on a cloud without an immediate

correspondent for database transactions.

The relatively simple coordination model discussed next is often used in cloud computing.