independIT Integrative Technologies GmbH
Bergstraße 6
D–86529 Schrobenhausen

schedulix!focus

Hierarchical Batch Structures

Dieter Stubler Ronald Jeninga

November 25, 2016

Hierarchical Batch Structures

Introduction

Process workflows in data warehouse environments frequently comprise a large number of work steps that have to be coordinated and started in the right sequence. Breaking down the individual steps as much as possible into sub-processes (process decomposition) allows the workflows and their progress to be monitored and controlled more effectively. When errors occur and processes are rerun, a fine granularity of the sub-processes allows errors to be remedied much more quickly and the hardware resources can be utilised more efficiently by reducing the 'lost' computing time.
Decomposing the workflows for a data warehouse into as many single processes as possible is therefore a key design goal for achieving and maintaining a stable and reliable data warehouse operation.
A persistent pursuit of this goal will quickly result in workflows with many hundred executable sub-processes.
If the deployed scheduling system does not support workflows with so many sub-processes, or only insufficiently, the advantages of process decomposition will be offset by the greater time and expense that has to be spent on implementation, monitoring and operation. This frequently means that limits are placed on the decomposition of sub-processes at a very early stage.

Hierarchical order of sub-processes

In order to be able to easily implement workflows comprising many sub-processes, and to avoid losing track of such workflows when monitoring and running them, the schedulix Scheduling System enables processes (jobs) to be grouped into hierarchical structures (batches).
Dependencies can be defined at any of these hierarchical levels to provide for the most simple, clearly organised and maintainable realisation possible of even the most complex workflow structures. Dependencies related to a sub-batch are only considered to have been fulfilled once all the jobs belonging to this sub-batch have been completed. Similarly, all the jobs in a sub-batch 'inherit' all the dependencies of the parent sub-batches.

Example: A simple workflow

A workflow is to create 3 reports (R1, R2 and R3). To do this, 3 database tables (T1, T2, T3) have to be updated first. Tables T1 and T2 can be updated in a parallel operation. T3 can only be updated once the update of T1 has been completed. The individual reports can be created parallel to one another. After the reports have been created, they are then to be published in a web server directory.

Assembling this workflow without a hierarchical order will make it look as shown in Figure
1 .

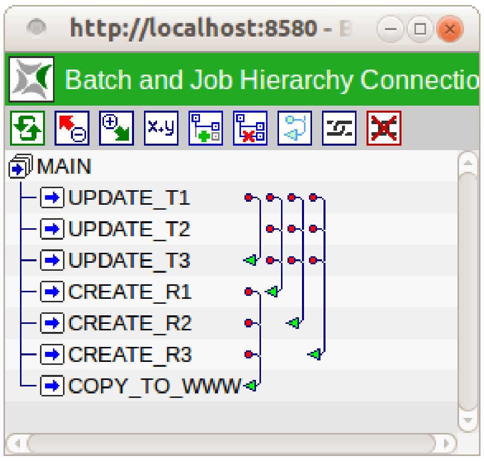


Figure 1: Batch structure without a hierarchical order

Even this relatively small workflow of 7 processes is not really transparent and requires the definition of 13 (!) process dependency relationships.
If an attempt is made to reduce the number of necessary dependencies without a hierarchical batch structure, this can be achieved by concatenating the single jobs. Figure 2 illustrates this solution. Although this looks somewhat clearer now, this approach still presents some problems.

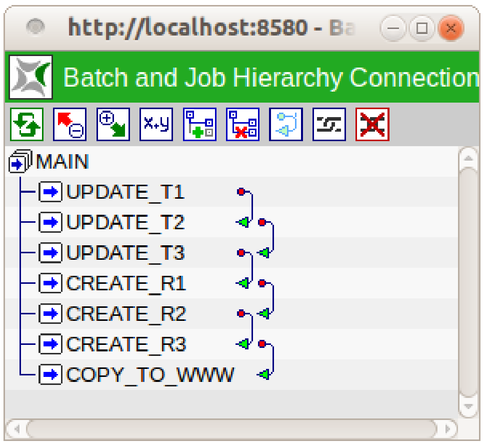
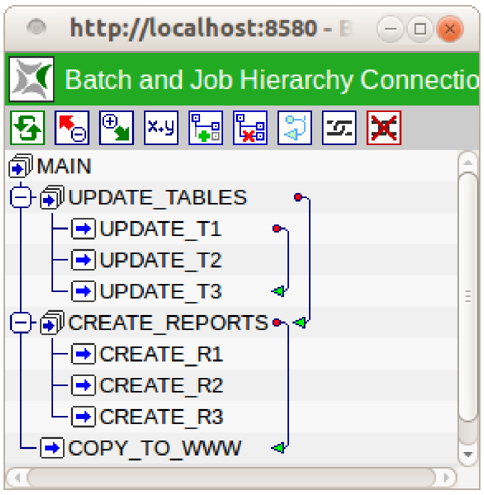


Figure 2: Batch structure as a chain

The individual processes now no longer run parallel to one another but consecutively in series, which means that the total time required to run the workflow is unnecessarily excessive.

The information about the existing dependencies is lost, maintainability is diminished and any optimisation potential disappears because a subsequent developer cannot decide whether the serial processing was technically necessary or easier to implement due to technical reasons.
This can be significantly simplified by using hierarchical batch structures. Figure 3 clearly illustrates this.



```
MAIN
  UPDATE_TABLES
    UPDATE_T1
    UPDATE_T2
    UPDATE_T3
  CREATE_REPORTS
    CREATE_R1
    CREATE_R2
    CREATE_R3
  COPY_TO_WWW
```

Figure 3: Batch structure with a hierarchical order

Grouping the UPDATE_...jobs and CREATE_...jobs into the (sub-) batches UPDATE_TABLES and CREATE_REPORTS makes the whole workflow construct much more manageable. The dependency of the CREATE_REPORTS batch on the UPDATE_TABLES batch is only considered to have been fulfilled once all the jobs in UPDATE_TABLES have been completed. The same applies to the dependency of the COPY_TO_WWW job on the CREATE_REPORTS batch. The CREATE_Rn jobs implicitly 'inherit' the dependency on the UPDATE_TABLES batch from their parent batch CREATE_REPORTS.
This means that now only 3 dependency relationships are required to run the jobs in the correct order.

Impact on maintainability

Hierarchical structures increase the maintainability of workflows by reducing the change requirements and the risk of errors.

Example: Integrating an additional process

To clarify this point, a fourth report R4 is to be created in our example and the requisite process is to be integrated in the workflow.

If this is done in the model without a hierarchical order, in addition to defining the CREATE_R4 job and attaching it to the MAIN batch, it is also necessary to define another 4 (!) extra dependencies to make sure that CREATE_R4 also waits for the jobs UPDATE_T1, UPDATE_T2 and UPDATE_T3 and that COPY_TO_WWW now also waits for the new job CREATE_R4. Figure 4 shows the new structure of our workflow.

It is very easy to overlook a dependency here, which would produce false results during the data warehouse operation. If we forget to make the COPY_TO_WWW job dependent on the new job CREATE_R4, this would mean that if CREATE_R4 were to run longer than the other CREATE_. . . jobs, the COPY_TO_WWW job would be triggered before the report R4 was created. This would result in the output of an incorrect or obsolete report. To make matters worse, this error does not always occur, but is dependent on the temporal behaviour of the processes involved, making troubleshooting much more difficult.

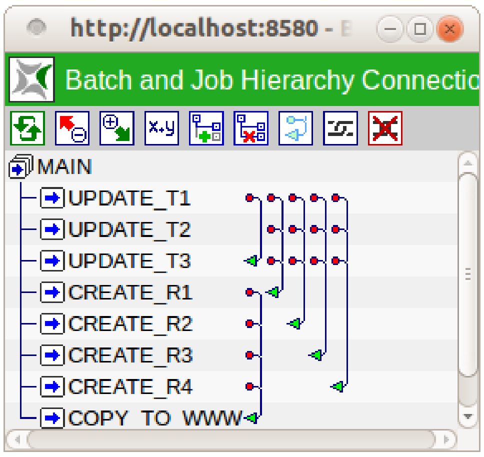
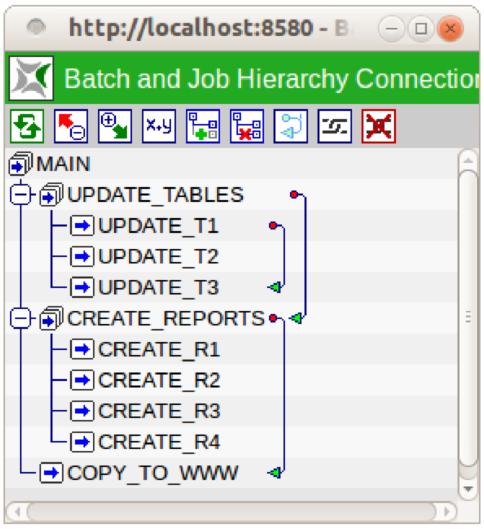


Figure 4: Batch structure without a hierarchical order after integrating the report R4

In our hierarchically organised workflow definition, it is only necessary to define the CREATE_R4 job and attach it to the CREATE_REPORTS sub-batch. No additional dependencies have to be created. The effort and risk of error are much diminished with this variant compared to the workflow without a hierarchical order. Figure 5 shows the new structure of our workflow.

Example: Breaking a process down into sub-processes (process decomposition)

The procedure for decomposing processes is also ideally supported by the hierarchical model of the schedulixScheduling System, as we shall demonstrate in the following example.



```
MAIN
  UPDATE_TABLES
    UPDATE_T1
    UPDATE_T2
    UPDATE_T3
  CREATE_REPORTS
    CREATE_R1
    CREATE_R2
    CREATE_R3
    CREATE_R4
  COPY_TO_WWW
```

Figure 5: Batch structure with a hierarchical order after integrating the report R4

The UPDATE_T1 process now runs for too long and is to be decomposed into two parallel executable processes UPDATE_T1_A and UPDATE_T1_B. In the schedulix Scheduling System this only requires the following actions:

1. The jobs UPDATE_T1_A and UPDATE_T1_B are defined.
2. The type is changed from UPDATE_T1 to 'BATCH'.
3. The jobs from the 1st are attached to the batch UPDATE_T1.

All the dependency relationships are preserved and the effort required to implement the change in the Scheduling System and the risk of errors are kept to a minimum. Figure 6 shows our workflow after the process decomposition. The CREATE_REPORTS sub-batch was closed in the illustration for reasons of clarity.
If this process decomposition were to be implemented in a workflow structure without a hierarchical order, in addition to the steps described here it would be necessary to create a large number of new dependency relationships to ensure that the jobs are run in the correct sequence.

Further properties of hierarchically structured workflows

Hierarchically organised workflow structures support other functions in addition to the properties mentioned here, but detailed descriptions are beyond the scope of this documentation.
For example:

- Reusability by means of parameterisation

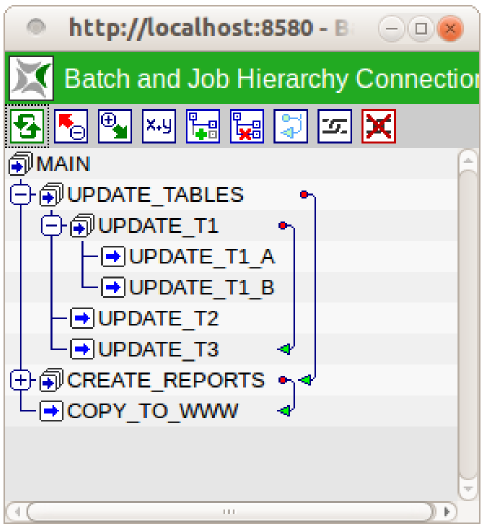


Figure 6: Batch structure following decomposition of the process UPDATE_T1 into parallel sub-processes

- Automatic error handling and notifications (triggers)
- Hierarchical operating (suspend, resume, rerun, cancel, . . . at (sub-) batch level)
- Conditional execution of sub-workflows
- Reruns of sub-workflows
- . . .

We will be taking a look at one or the other aspect in a future schedulix!focus.

Closing remarks

Even the extremely simple examples in this documentation clearly demonstrate the advantages of hierarchical batch structures when modelling workflows. In larger workflows with a huge number of sub-processes, this functionality of the utilised scheduling system becomes a necessity without which an optimised and practical process decomposition would no longer be possible.

The schedulix Scheduling System, with its highly developed and powerful concept for the hierarchical mapping of processes, ideally supports process decomposition while preserving the clarity and maintainability of even extremely large and complex workflow structures.