December 2021

A METHOD TO AGGREGATE 3D JOB INFORMATION

HP INC

A method to aggregate 3D job information

Abstract

MJF Printers build a 3D job on a Build Unit storing build metadata information and a low-resolution image to be able to show the build information when the Build Unit is moved to next device, for example a Processing Station. However, when inserting the Build Unit into the next device the information might take some time to be read and be ready to operate.
Additionally, build might be transferred to a Cooling Unit with less storage capabilities, causing to store very limited information, which might not be enough for some operations.
The job new job orchestration would require having a centralize repository to distribute the job information when needed by the devices.
This document describes a multi-level information approach to have as early as possible some minimum information to show about the build, and an aggregation mechanism to incrementally add the required information, like low resolution image, detailed content, and instruction on how to proceed from different sources.

Introduction

A Job in the HP MJF devices (printers, processing, stations, unpacking units, …) contain a rich set of information that describes:

- Job information provided by the job creator or submitter, such job name, application, or user generating the job, etc.
- What to build, as information extracted from the build description package, i.e., 3MF, containing geometry information, parts to print, as well as images for the whole build or the individual parts to show in the FP.
- Information on what operations to perform, or input ticket.
- Information captured while executing the operation, stored in the output ticket. It would also contain images for each layer generated. As well as how it was printed, and supplies consumption.

This is information is stored as an information model, and could be stored serialized to Json, XML, etc, plus a set of images files produced for build, parts, layers.

An external SW orchestrates the distribution of the different operations across the devices, orchestrating what to perform. This orchestration SW would collect the resultant information model from the execution of an operation and make it available to next devices.

On other occasions, there is no orchestration SW, and a subset of information model is stored in the trolleys (build unit or cooling unit) and transferred to the next device by reading from its storage. However, the information they can store is limited in size due to storage restrictions. For example, a build unit can store input and output tickets, and a low-resolution build image, but the whole list of

parts, neither their thumbnail images. A cooling unit is even more restricted to be able to store just a few parameters (a job summary), like job id, name, material id, cooling information, number of parts, and just a few more.

When a trolley (BU, CU) is inserted into a new device, it identifies what build job is in the trolley to show the information onto the FP and be ready to execute next operations.

Proposed method

The proposal is then when inserting the trolley containing a job in a new device, reconstructing the information model happen in several steps, as shown in *Figure 1*, to get enough job information to show to the user on the FP, and to perform the requested operations.

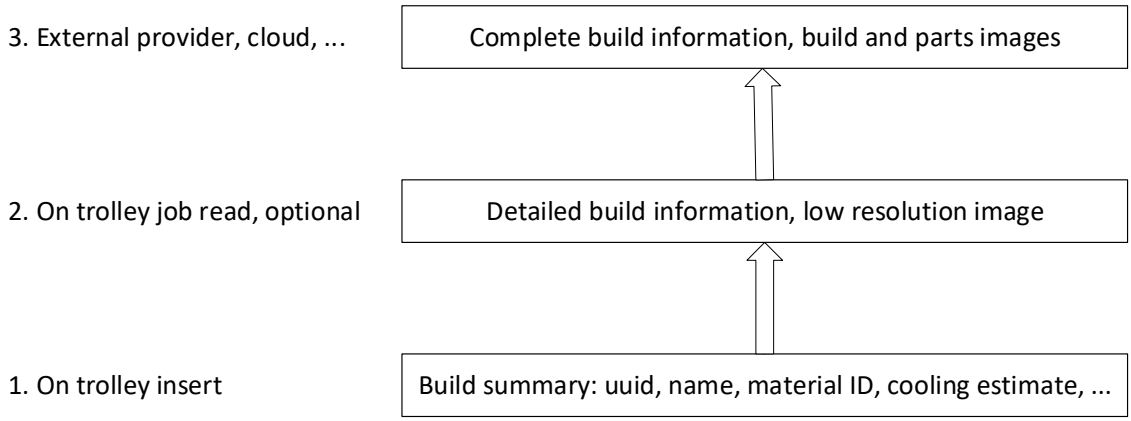

Figure 1. Job information aggregation sequence

Step 1:

Reading a job summary happens immediately but has very limited information to show. Also, the small set of parameters might limit the operation that could be executed. For example, it might cool the build, but nothing else.

With this amount of information model, the device could show some basic information about the build job, like job name, number of parts, etc. to the FP. But not any image about the build.

Some operation of the build might be enabled, but other operations that require more detailed parameters might not be enabled.

In previous implementation, the build unit did not have this summary, and had to skip providing the information to the FP until next step.

Step 2:

Once the optional build payload in the trolley storage is available, after several seconds (up to more than one minute), more detailed job information model and a build low resolution image is read from the trolley storage to be aggregated.

At that time the FP can show more build job information including the build image.

The ticket included in the information model would provide instructions on what to execute. But it might not contain very detail level of information, like parts information or images that require next step.
On pure job orchestrated environments this step might be omitted, not stored in the trolley, like in a cooling unit. This requires that the job information, build image, etc., would be provided in next step. However, in mixed environments, or when there is no guaranty that the orchestration SW is available all the time, this helps to provide a minimum information to operate. This is the case with current MJF.

Step 3:

Detailed job information, uploaded from previous devices executing operations to the build job, is stored externally to the device, for example in the cloud. This information could contain full detail information model, not only for the build, but also for the parts, including images of each part. It would contain the logging as output ticket from the execution on the different devices.
Providing that information to the device might be either by the device requesting it through an event, or the arbitration SW periodically polling the device for jobs to update.
When updating the job model information available, with the one provided externally, it must merge both information models to have them consistent. The rule to follow here is that any information generated on other devices (for example the input ticket, or any output ticket with operations from other devices) is updated with the incoming information model, overriding the existing one. And any information generated internally in the device, like the output ticket with operations from the device, stays, since the device is the one providing it.

All this system is complemented with a job history database to store the job information models. This way, if a trolley is inserted again, the information would be already be available. However, it might require updating (merging) with information model coming from executions by other devices.

Disclosed by Jordi Gonzalez Rogel, Yngvar Rossow Sethne, Jordi Amoros and Guiu Tio, HP Inc.