


Online Workflow Puzzle

ABSTRACT

We are presenting an online workflow puzzle, which runs in a browser. The player can put together puzzle pieces to pre-defined production chains. Extensive online help supports the player in laying out the puzzle pieces correctly. The puzzle is based on the process-resource model, which is also the basis of the Job Definition Format. This paper explains the game as well as implementation strategy.

Thomas Hoffman-Walbeck¹ 
Richard Adams²

¹ Stuttgart Media University,
Stuttgart, Germany

² Ryerson University,
Toronto, Canada

*Corresponding author:
Thomas Hoffman-Walbeck
e-mail: hoffmann@hdm-stuttgart.de*

KEY WORDS

workflow, online game, JavaScript, HTML, jQuery, JDF

First received: 1.6.2020.

Revised: 4.9.2020.

Accepted: 7.9.2020.

Introduction

A Print Service Provider (PSP) needs many experts in order to accomplish all the different production processes, which are necessary for manufacturing a print product. Those experts usually have a deep knowledge in their special fields. For example, to determine the width of the spine for a hard cover book sounds easy at the first glance, however, it often requires a lot of expertise and experience. On the other hands, project and production manager do not need to know all details of each production process. They should be able to pre-define the necessary production steps, their sequence, their upfront requirements and their outcomes. This reflects the old mantra of specialists and generalists.

Conducting student projects in graphical departments at universities, we noticed that even students tend to specialize very much. Some of them are interested in product design (only), others in Prepress, Press or Postpress. That is, in practical group proj-

ects most students like to pick their favorite topic, which they know best already. In the end, it is hard for most of them, to outline a production overview.

Several years ago, we created different versions of paper-based gaming cards for different processes: In lab classes, the students laid the cards on a table in an ordered manner displaying a specific production workflow. The students paired up in small groups in order to stimulate a discussion about the solution.

Increasing production automation definitely is key in the Graphical Industry. There are two very different ways to accomplish that:

- a. automating individual processes or
- b. automating the workflow

Automation of machines falls into category a). That is, robots, cobots and production equipment that, for example, shortens make-ready times.

Software algorithm in Prepress for automating an individual task we also subsume under a).

Automation of class b) has two variants. One is of purely organizational (e.g. by determining who is supposed to transmit at what time imaged plates to the offset press); the other one has more of an IT aspect: Devices and software modules sharing information to make sure that each process gets all the required resources that they need for executing in time. There are still many optimization potentials in this. It is the area of the Job Definition Format (CIP4, 2020a), where product descriptions and production details are sent to several devices so that those can use them for an (automatic) execution of a process. Often, this is called an “integration” of processes.

JDF employs the process-resource model for the product description. This is also true for XJDF (CIP4, 2020b), but to a somewhat lesser extent. A process is an activity like printing or folding. Mostly, a process requires input resources for execution and generates one or more output resources. In general, a resource is either some physical object (like plates or paper) or some electronic/conceptual entities (like PDF pages, parameter sets). Some resources are output resources of some process and in the same time input resources for some other. We are calling those “transactional” resources. An images plate, for example, is the output resource of the process “image setting” and in the same time an input resource of the “printing” process. A physical press would be an example for a non-transactional input resource of the printing process. Note, that the transactional resources imply the sequence of processes. Thus, they are important for scheduling.

In summary, we would like to justify our motivation for a design a workflow puzzle that on the one hand, we consider the integration of technical processes in print production extremely important for the graphical industry and on the other hand, we see that students have a certain lack of knowledge with this respect. Getting an overview of production sequences is the first right step in this direction.

Our puzzle game is based on the process-resource model. The player can lay down puzzle pieces that represent either processes or resources. Such a puzzle pieces is also called a “card” in this paper and in the code of our game. Since in a real production environment the model can get complicated with dozens of processes and hundreds of resources, we had to simplify the model. We archived that by focusing on transactional resources.

We need to clarify that the names of the processes and the resources that we are using in the game, are not necessarily the names that CIP4 specified in the JDF specification. The latter are frequently some-

what abstract (like “RunList” or “Component”) and a bit hard to comprehend for students. We also like to mention, that the process-resource model is not limited to a model for print production only, but suits any workflow. This holds for our puzzle as well.

We believe that this re-design of the analogue card game to an online puzzle game has several advantages for the students:

- They have access to online-help about processes and resources if they need more information about them
- Since they do not need a physical card game, they can play the puzzle outside the class
- They get an immediate respond if the lay the card in a wrong order

How to play the puzzle

We hosted the puzzle game under:

<https://www.ryerson.ca/~wdp/workflow-game/>

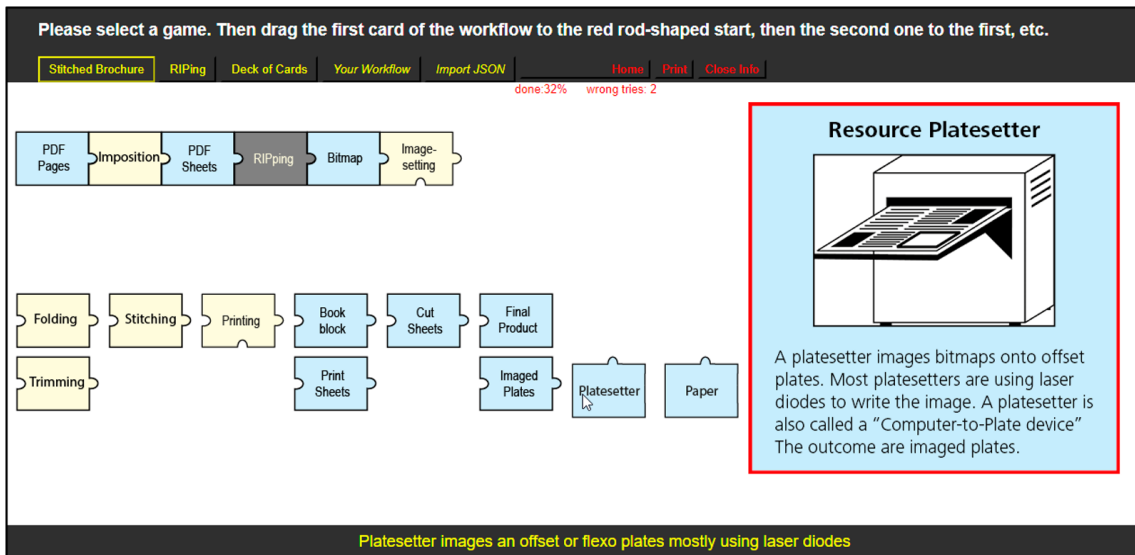
We encourage you to try it yourself.

With the puzzle, we are pursuing two different didactic goals. In the first case, we like to teach the sequence of different tasks for a certain workflow. Here we marked the cards either by a number or by some other strong hint how to place the cards, e.g. by some color scheme. In the second case, we are asking the player to do his or her own research about the sequence of processes and resources. For that, there is information attached to each card. If the player hovers over a card, he or she will see a short statement at the footer about what the card represents in a yellow typeface. If that is not enough, a double click opens a window displaying a longer explanation in a pop up (framed in red color) giving some hints concerning the previous and next cards.

A counter counts the successful placements of cards as well as the number of failed attempts. We would consider this player best, who got the lowest number of wrong tries.

All yellow cards in Figure 1 represent processes, the gray one a group of processes (Interpreting, Rendering, and Screening) and all blue ones resources. The six card in the top row have already been moved to the correct position, the cards on the lower part have not. The small red information on the white background is saying that 32% of the cards are on the right position and two attempts for moving a card failed (because it was the wrong card).

The player receives some feedback when he or she lays out a card. If it is correct, a tone sounds and a textual confirmation comes up. If the card is



» **Figure 1:** Puzzle "Stitched brochure"

wrong, there is either a special indication of why the card is wrong or just a general objection.

Figure 2 shows another example of a puzzle game with a different card design. Here, only a small part of the overall production workflow is modeled, i.e. the RIPping process group.



» **Figure 2:** Puzzle RIPping

Tools

We wrote the puzzle game in HTML 5 (W3C, 2017) and JavaScript (w3schools.com, 2020a). We deployed the JavaScript libraries jQuery version 3.5.1 and jQuery UI version 1.12.1 (The jQuery Foundation, 2020). Editing the code, we used Brackets 1.14 (Adobe, 2020). For testing and debugging the HTML and JavaScript, we loaded the code into Google Chrome (Version 83.0.4103.61) and Mozilla Firefox (version 76.0.1). As a web server is necessary for the test, we installed XAMPP, version 7.4.6 for Windows, (Apache Friends, 2020) locally, in particular the Apache HTTP Server. All of these tools can be downloaded free of charge.

Implementation

We defined a class Card to store the relevant data for each card like the file name, the dimensions, the

neighboring cards and the HTML object (see Figure 3). The data we either defined directly in the script, e.g.

```
c301= new Card('card301','card301.png',
93,61,'card302',null,null'text info');
```

or by an import of a JSON file. For the specification of this data interchange format see (ECMA, 2017), for an easy online learning (w3schools.com, 2020b). Figure 4 shows an excerpt of our JSON file. The import works via HTTP. The jQuery method getJSON() reads the JSON data into the script. The so-called response function results in (key,value) tuples that need to be mapped to all Card object. Please note that the method getJSON() normally runs in the background (*asynchronous*), i.e. it take a while until the data is read in. Since the script should not continue in our case, before all data is available (otherwise we get many *undefined* errors), we need to switch to the synchronous modus beforehand by

```
$.ajaxSetup({ async: false});
```

Ajax is an acronym for Asynchronous JavaScript and XML.

For each Card object, we are creating dynamically an HTML element <div>. Using the jQuery function addClass() and innerHTML, we can complete the element by defining a class for it as well as the usual HTML sub-element with property src specifying the path to the image file. These elements we have to append to the HTML Document Object Model (DOM) structure. Moreover, we added the HTML objects to the Card structure – see objectHTML in Figure 3. All Card objects of a game are pooled in an array that we called cardObjects. With this array, we have all necessary information for all cards ready to lay them down on the window.

For moving the cards is very easy to implement, because it is a given jQuery method called draggable.

```
class Card {
  constructor(id, scr, width,height,
    neighRight,neighBottom,neighLeft
    info,objectHTML){
    this.id =id;
    this.scr=scr;
    this.width=width;
    this.height=height;
    this.neighRight=neighRight;
    this.neighBottom=neighBottom;
    this.neighLeft=neighLeft;
    this.info = info;
    this.objectHTML = objectHTML;
  }
  ...
}
```

» **Figure 3:** Class Card hold information for each card

```
[
  {
    "id":"card201",
    "scr":"card201.png",
    "width":93,
    "height":61,
    "neighRight":"card202",
    "neighBottom":"null",
    "neighLeft":"null",
    "helpText":"Create cards using a
      graphic editor"
  },
  ...
]
```

» **Figure 4:** JSON data of a Card object

A card is supposed to snap to the correct position if it is close to its predecessor card, which in turn has been put down already correctly to the process/resource structure. To archive that, we have to check two things

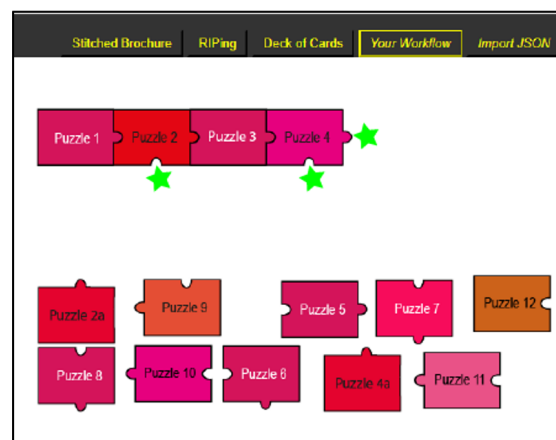
- i. Is the card close to a card, which needs a neighbor?
- ii. Is it the correct card in the process/resource chain?

For evaluating situation in i) we defined “hotspots”. A hotspot is an (x, y) position to which another card can be placed. Figure 5 shows the constructor of our class hotspot. Since our workflows need not to be strictly linear but rather allow ramifications, we might have several hotspots in the same time. In Figure 6, we marked each hotspot with a green asterisk. Therefor we defined an array of hotspot objects. Each card, which the player drags to some place, has be checked if it is close to one of those hotspots. In each hotSpot, information is stored, which is the correct neighbor. The trickiest part of the script, however, is the positioning of the card.

Depending if the predecessor card has an output “knob” or not, the card must me positioned differently. There are more situations like that which needs extra care.

```
class hotSpot{
  constructor(x,y,nextCardRight,
    nextCardBottom, nextCardLeft
  {
    this.x = x;
    this.y = y;
    this.nextCardRight = nextCardRight;
    this.nextCardBottom= nextCardBottom;
    this.nextCardLeft = nextCardLeft;
  }
  ...
}
```

» **Figure 5:** Constructor of the class hotSpot



» **Figure 6:** Each hotspot is marked with an asterisk

Conclusion

The Online Workflow Puzzle illustrates the trends toward “gamification” and asynchronous learning in the online environment. Details on this topic can be found in (Kim et al., 2018). As compared with the original printed “card deck,” the online version enables a wider audience of graphic arts students to explore various workflow steps and understand the connection between them. The configurable nature of the puzzle enables it to evolve with new technologies and the workflows in which they are used.

Many details are still missing in this puzzle game, like a

- cards with more than one inter-faces for output resources,
- support for more than one starting point,
- processing more than one JSON file with an automatic extension of workflow choice in the UI when importing new JSON data.

Finally, the game could have a completely different architecture in order to support “arbitrary” work-flows and not just predefined ones. That is, the front end (web browser) could have access to a superset of cards in unlimited numbers. The cards are dynamically generated by the software according to the player’s choice of inputs and outputs for all four edges. The production configuration, which the player constructs, is sent to a backend software for evaluation. The backend software works with a knowledge base, consisting of rules that can be edited independently. This way, configurations that contradict one or more rules (as the printing process is followed by the imposition process) could be detected and warning could be forwarded to the frontend and ultimately to the player.

Kim, S., Song, K., Lockee, B. & Burton, J. (2018) Gamification in Learning and Education. Cham, Springer. Available from: doi: 10.1007/978-3-319-47283-6

The jQuery Foundation (2020) jQuery 3.5.1. Available from: <https://jquery.com/> [Accessed 28th August 2020].

W3C (2017) HTML 5.2. Available from: <https://www.w3.org/TR/html52/> [Accessed 28th August 2020].

w3schools.com (2020a) JavaScript Tutorial. Available from: <https://www.w3schools.com/js/DEFAULT.asp> [Accessed 28th August 2020].

w3schools.com (2020b) JSON – Introduction. Available from: https://www.w3schools.com/js/js_json_intro.asp [Accessed 28th August 2020].

References

Adobe (2020) Brackets. Available from: <http://brackets.io/> [Accessed 28th August 2020].

Apache Friends (2020) XAMPP. Available from: <https://www.apachefriends.org/de/index.html> [Accessed 28th August 2020].

CIP4 (2020a) JDF Specification 1.7 final. Available from: <https://confluence.cip4.org/display/PUB/JDF> [Accessed 28th August 2020].

CIP4 (2020b) XJDF 2.1 final. Available from: <https://confluence.cip4.org/display/PUB/XJDF> [Accessed 28th August 2020].

ECMA (2017) The JSON Data Interchange Syntax. 2nd ed. Geneva, ECMA International. Available from: <https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf> [Accessed 28th August 2020].



© 2021 Authors. Published by the University of Novi Sad, Faculty of Technical Sciences, Department of Graphic Engineering and Design. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution license 3.0 Serbia (<http://creativecommons.org/licenses/by/3.0/rs/>).