https://doi.org/10.24867/GRID-2020-p70 Professional paper

# VISUALIZATION OF THE PROCESS-RESOURCE MODEL FOR WORKFLOWS

*Thomas Hoffman-Walbeck*[1], *Richard Adams II*[2]
[1] *Stuttgart Media University, Stuttgart, Germany*
[2] *Ryerson University, Toronto, Canada*

Abstract: *We describe the design of a JavaScript prototype that allows the user to visualize an arbitrary production workflow for print production that is based on the Process-Resource Model. The user can create square cards of type "process," "resource" or "resource group." For each card, a name might be chosen out of a list and neighboring relations can be set (input and output). Each card should be positioned on a fixed grid. Finally, the user can check the workflow design according to different reasoning.*

Key words: Workflow, Process Resource Model, Visualization, JavaScript, JDF

## 1. INTRODUCTION

For the graphic arts industry, automation is the key to reducing human touchpoints in the production process. In very general terms, this can be done by optimizing the organization and improving devices (machines or software modules) that handle one or more production processes. Device optimizations are achieved either by shortening the presetting time or by automating the actual production phase.
Especially for presetting, a device needs data that is generated outside of the device. For example, a printing press needs to know the type, quality, thickness, and dimension of the printing substrates, the primary inks and the printing sequence and so on. Of course, the press also needs physical assets like the actual paper, ink, and printing forms (plates) for conventional printing. Both these physical assets and the data we subsume under the notion "resources." That is, the process "printing" needs input resources, then it generates an output resource (print sheets), which in turn will be inputted by the next process. A "Process-Resource Model" (PRM) is a description of (production) steps via processes and resources in a net topology.
"Workflow automation" denotes the method of providing input resources for processes automatically as well as importing theses resources into a device so that devices can react on them in a self-acting manner. The data formats JDF/JMF (CIP4, 2018a) and XJDF/XJMF (CIP4, 2018b) are based on the PRM.
As a user, one does not pay attention to these formats, just as little as the underlying PRM. For the planning of new production methods or production lines, however, an understanding of PRM is essential. In order to communicate a PRM (e.g., between Print Provider and system integrator), a visualization of PRMs are necessary. Since there is no standard method and no tool of visualizing a PRM, we developed a JavaScript-based interactive tool called "Workflow Arrows."

## 2. HOW TO USE THE SCRIPT

We called the script "Workflow Arrows," because input and output resources are linked to processes by arrows. We hosted the script under:

*https://www.ryerson.ca/~wdp/workflow-puzzle/print/Test/Workflow_Arrows.html.*

We encourage you to try it out yourself.
Figure 1 shows the user interface of the script as well as an example of a workflow description for a hardcover book production.
In "Card Type," you can define whether the card should represent a process (yellow), a resource (blue), or process group (gray). Process groups are collections of processes, which can be handled in the tool like a process. The difference is that a process group contains several processes or – in other words – they represent a wider range of the print production, like "Prepress Preps."
Please note, that a resource cannot generate a process and similarly a process cannot generate a process. That is, a processes and resources always have to alternate.

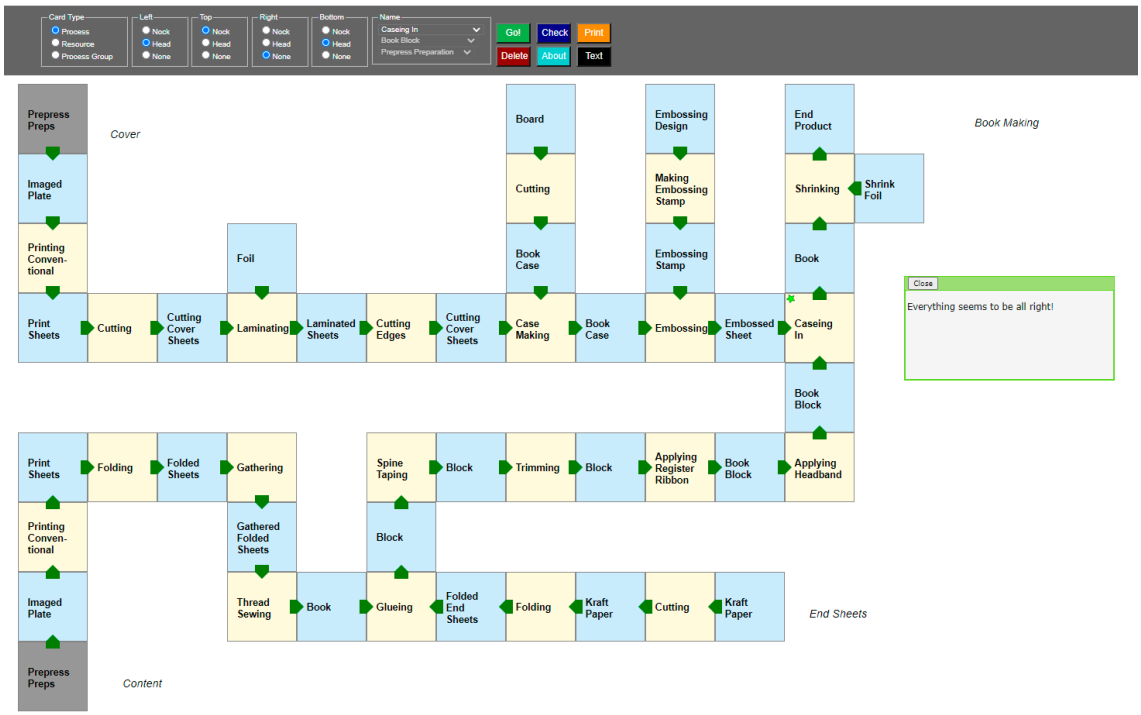


*Figure 1: Process-Resource model for the hardcover book production. The small green dialog box on the right shows the result of the "check".*

Arrows indicate the connection of resources and processes. Each arrow consists of two parts: The arrow nock (a rectangle) and the arrowhead (a triangle). Each arrow nock needs an arrowhead as a counterpart. For example, in Figure 2 the process card "Folding" has an arrow nock on the right-hand edge, while the resource card "Folded Sheets" has an arrow head on the left-hand edge. For each edge of a card, you may create arrow nocks or arrowheads. You also should choose a specific name of a process, group, or resource from the list.

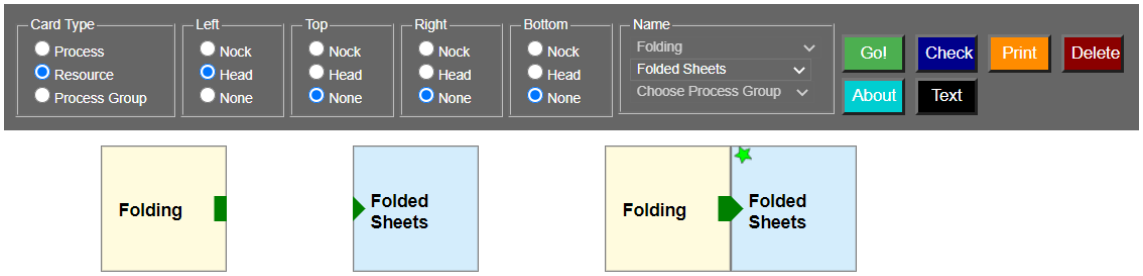


*Figure 2: An arrow nock and an arrowhead compose an arrow. On the right-hand side, the two cards are next to each other – building a proper arrow.*

After specifying the type, the arrows and the name of a card, clicking on the button "go" will create the HTML element. Next, you need to move the card to its approximate position. It will snap onto an invisible grid. A little green star on the upper left corner of the card indicates the last card you moved to the grid (or the last selected card).
If the user cannot find the card name in the list, she or he can create an own name by clicking on **Own Process**, **Own Resource**, or **Own Process Group** in selection list "Name." In Figure 3, the card "Wand" and "Perform Magic" are generated that way.

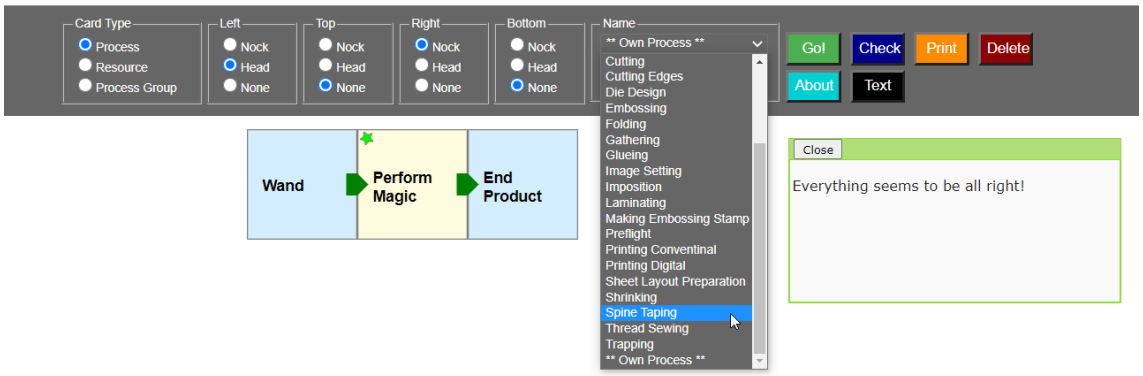


*Figure 3: Names of processes, resources or process groups can be defined arbitrarily*

Each PRM needs a start and an end. They can be either a process card or a resource card. A start card may not have any input; the end card may not have any output. Typically, the end card represents the final product.

A workflow can be checked by clicking on the button “check.” However, only the formal structure of the PRM will be analyzed. For example, it will be checked, if there are at least one start card and one end card available and whether the structure follows the PRM requirements, e.g., alternating processes and resources. However, the user is responsible for laying out the right processes and resources for the production. If no error is found, there will be the message “Everything seems to be all right!” (See Figure 3). If the script finds a mistake, it will be describes accordingly as to be seen in Figure 4.

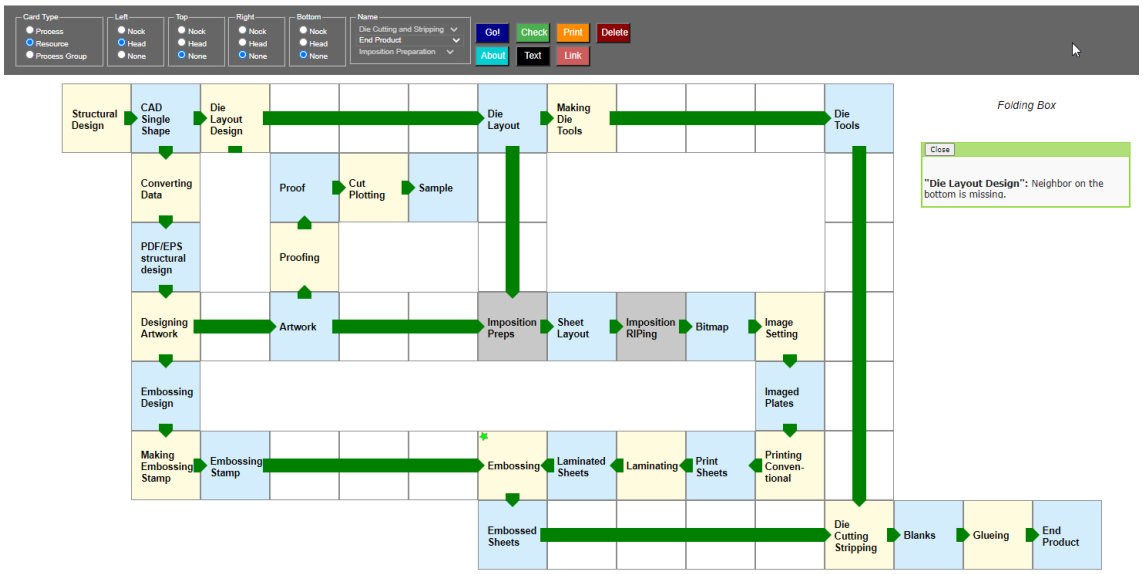


*Figure 4: The Workflow describes the production of folding boxes. Please note that the check found an error. There is a mistake in the card “Die Layout Design”, which erroneously has an arrow nock at the bottom edge.*

In a more complex workflow like shown in Figure 4, it is necessary to combine cards that are not direct neighbors. This can be achieved by using the button “link.” Before that the edges have to be specified, whereas is does not make any difference if you click on “nock” or “head.”

## 3. TOOLS

We wrote the tool in HTML 5 (W3C, 2017) and JavaScript (W3C, 2020). We deployed the JavaScript libraries jQuery (version 3.5.1) and jQuery UI (version 1.12.1) (JQuery, 2020). Editing the code, we used Brackets (Release 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). All of these tools can be downloaded free of charge.

## 4. IMPLEMENTATION

We were using standard JavaScript and jQuery to build the UI. The content of the cards are described in SVG (W3C, 2011), according to the setting of “Card Type”, “Name”, “Left”, “Top”, “Right” and “Bottom” in the UI.
In the script, we defined two classes, “Card” and “Cards”. With the first one, we can define objects for each card, incorporating the following properties: “card Type”, “Name”, “Left”, “Top”, “Right”, “Bottom”, position (row/column on the grid), id of the HTML element and a Boolean variable, weather the element has been deleted or not.
The instance of class “Cards” incorporates an array of all card objects, that is, it defines the entire topology of the net.
The configuration of the cards can be stored in the local browser employing the web storage API. The data is stored as properties of the storage object, i.e. as key/value pairs. Of course, the data can also be retrieved from there.

## 5. DISCUSSION

Many small enhancements are needed before this prototype could become an application. It would be more challenging, however, to add certain production requirements to be tested concerning rules like “Image Setting must be followed by Imaged Plate,” “Imaged Plate must be followed by Conventional Printing,” Image Setting is a Prepress process,” “folding is a Postpress process,” “Prepress process must not be behind a Postpress process” and so on. Possibly, AI method could be employed.

## 6. REFERENCES


[1] Adobe (2020): "Brackets, version 1.14.2", URL: http://brackets.io/ (last request: 2020-08-15.)
[2] CIP4 (2018a): "Cooperation for the Integration of Processes in Prepress, Press, and Postpress (CIP4): “JDF Specification 1.6-final (2018)", URL: https://confluence.cip4.org/display/PUB/JDF (last request: 2020-08-14.).
[3] CIP4 (2018b): "Cooperation for the Integration of Processes in Prepress, Press, and Postpress (CIP4): “XJDF 2.0 (2018) ", URL: https://confluence.cip4.org/display/PUB/XJDF (last request: 2020-08-14.).
[4] JQuery (2020): "Open JS Foundation (Linux Foundation Projects): jQuery 3.5.1", URL: https://jquery.com/ (last request: 2020-08-15.)
[5] W3C (2017): "HTML 5.2 (2017)", URL: https://www.w3.org/TR/html52/ (last request: 2020-08-15.)
[6] W3C (2020): "JavaScript Tutorial", URL: https://www.w3schools.com/js/DEFAULT.asp (last request: 2020-08-16.)
[7] W3C (2011): "Scalable Vector Graphics (SVG) 1.1, Second Edition, 2011", URL: https://www.w3.org/TR/SVG11/ (last request: 2020-08-15.)