

# A Resource Manager for Advanced Resource Management and Allocation in Processes

Sven Ihde<sup>1</sup>, Maximilian Völker<sup>1</sup>, Luise Pufahl<sup>2</sup> and Mathias Weske<sup>1</sup>

<sup>1</sup>Hasso Plattner Institut, University of Potsdam, Potsdam, Germany

<sup>2</sup>Software & Business Engineering, Technische Universität Berlin, Berlin, Germany

## Abstract

Resources play an essential role in the execution of business processes as they perform the work of the business steps. Thus, resource management and allocation has a high impact on the effectiveness and efficiency of the business processes. In existing process execution systems, the capabilities of managing and allocating resources are limited. In this demo, we present a resource manager, Rembrandt, that supports to specify resources with the help of hierarchies and attributes, as well as resource allocation recipes. The allocation recipes can also be executed, whereby advanced allocation algorithms of any programming languages can be integrated. Rembrandt offers a front-end and REST APIs that can be used by practitioners and researchers to apply its capabilities in process execution systems or simulators.

## 1. Introduction

Resources are essential for the process execution (human and non-human) ensuring the process progress by executing the tasks of a process case. Often resources are limited and expensive such that companies strive to optimize their resource utilization whenever possible [1]. Traditional process execution engines, such as Camunda<sup>1</sup>, offer only limited techniques for resource management, where resources are grouped into roles and allocated in a simple rule-based manner. However, rule-based approaches can lead to sub-optimal solutions [2]. An effective and efficient process execution needs more complex resource management with a problem-oriented resource allocation [1, 3].

In this demo, we present an open-source resource manager, called Rembrandt<sup>2</sup>, enhancing process execution engines or process simulators by advanced resource management and allocation capabilities. Rembrandt enables users, on the one hand, to represent the organization's resources and resource hierarchies and specify their attributes. On the other hand, resource allocation plans (i.e. allocation recipes) can be defined where the input, all available resources, and the output, the allocated resources, can be specified. For running these plans, one or


---

*Proceedings of the Demonstration & Resources Track, Best BPM Dissertation Award, and Doctoral Consortium at BPM 2021 co-located with the 19th International Conference on Business Process Management, BPM 2021, Rome, Italy, September 6-10, 2021*

✉ sven.ihde@hpi.de (S. Ihde); maximilian.voelker@hpi.de (M. Völker); luise.pufahl@tu-berlin.de (L. Pufahl); mathias.weske@hpi.de (M. Weske)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

<sup>1</sup><https://github.com/camunda/camunda-bpm-platform>

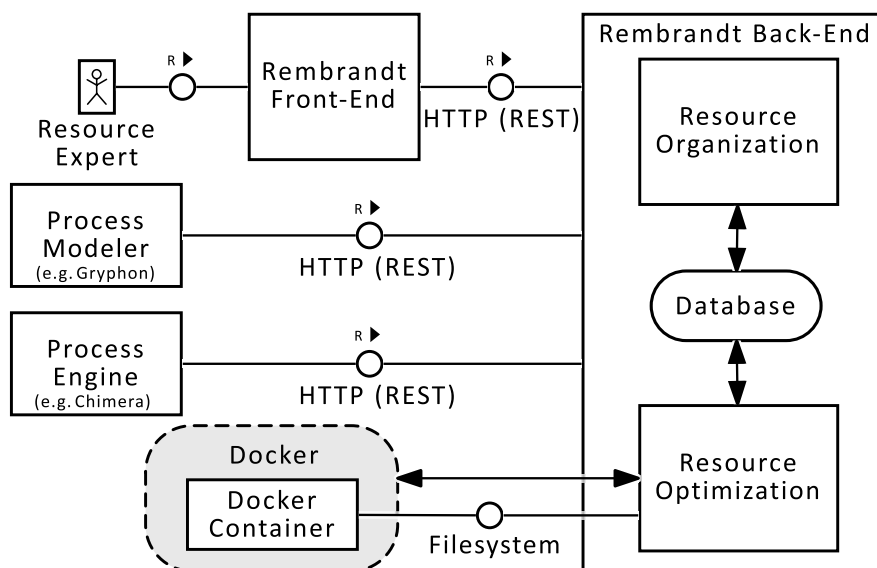
<sup>2</sup>Code: <https://github.com/bptlab/rembrandt>

Documentation including Screencast: <https://rembrandt.gitbook.io/docs/>

several techniques or heuristics (e.g., the Munkres algorithm [4]) from other domains, such as operations management, can be specified by the user. They can be specified in any coding language, as Rembrandt uses docker<sup>3</sup> for executing the allocation algorithms.

In the remainder, we present the main functionalities in more detail. To show the maturity of the approach, we demonstrate two integrations of the tool.

## 2. Main functionality



**Figure 1:** General architecture of the resource manager Rembrandt. Additionally showing possible integrations with external tools

The resource manager Rembrandt can be used to create, store, and delete resources and is able to integrate optimization algorithms to use them for the advanced allocation of resources. The back-end consists of two components as shown in the architecture in Fig. 1, the resource organization and the resource optimization. The resource manager provides different interfaces to connect to, a user interface for human operators and a REST API for other systems, such as Camunda.

### 2.1. Resource Organization

In this component, resource types and instances can be defined, either via the Rembrandt front-end or the provided REST API.

Rembrandt allows to define hierarchies between resource types. As shown in Fig. 2, Letter and Parcel can be modelled as sub-types of a Shipment type, and therefore share the attributes *receiver*, *sender*, and *delivered*, but the attributes *dimensions* and *weight* are exclusive to parcels.

<sup>3</sup><https://www.docker.com/>

## 2.2. Resource Optimization

In this component, resource recipes can be defined, that are executed during run-time via the provided REST API to receive an optimized resource allocation for a process activity.

**Allocation recipes.** Resource allocations first need to define the resources required as well as additional constraints. This can be specified via so-called allocation recipes in Rembrandt. They contain step-by-step instructions executed by the resource optimization component as soon as the corresponding optimization is requested. Rembrandt offers in its front-end a user interface for the creation of recipes as depicted in Fig. 3. It is a block based plug-in system with four different types of basic blocks, called ingredients: Ingredients are independent parts of the recipe that are self-contained, reusable components. Each recipe starts with at least one input ingredient, which is one of the predefined resource types. They are used to gather all resource instances of this specific resource type. In front and behind of optimization algorithms, transformer ingredients can be used. They offer functionality to modify resources to fit the respective needs of the following building block. Transformers can change values of resource attributes, filter the list of resources, or even change the type of the object they are working on. The most important ingredient is the optimization ingredient. It symbolizes a concrete optimization algorithm created by the optimization expert and has an arbitrary number of input ports and output ports, depending on the number of defined inputs and outputs when registering the algorithm to the platform via a docker container. At the end of each recipe an output ingredient is placed which contains the result of the optimization. Each output ingredient saves its result in the resource manager as one resource instance of the respective type so that the platform can continue working with it. Furthermore, recipes can be reused within other recipes to build upon previous work and merge common steps into one block. This allows combining multiple smaller solutions to one complex optimization pipeline, without loss of structure and clarity.

**Resource Allocation** Resource allocation is responsible for taking resource allocation requests, e.g., from a process execution system, collecting them, and executing recipes inside the resource manager. A resource allocation requests describes the requested type of resource(s) and the needed allocation recipe. During execution, all resource instances of all input resource types for this recipe are gathered first. Secondly, transformers are applied to the collected resource instances and subsequently, their results are passed to the algorithm which is executed using the respective docker container. Lastly, the output of it is stored as a new or modified resource instance and returned to the requestor.

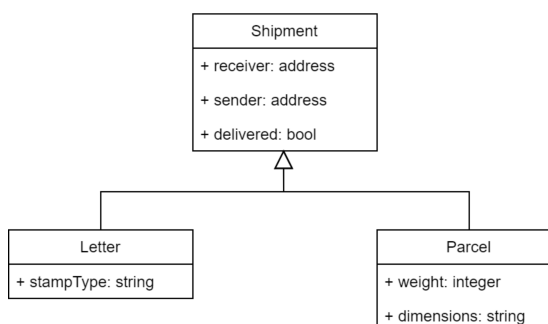
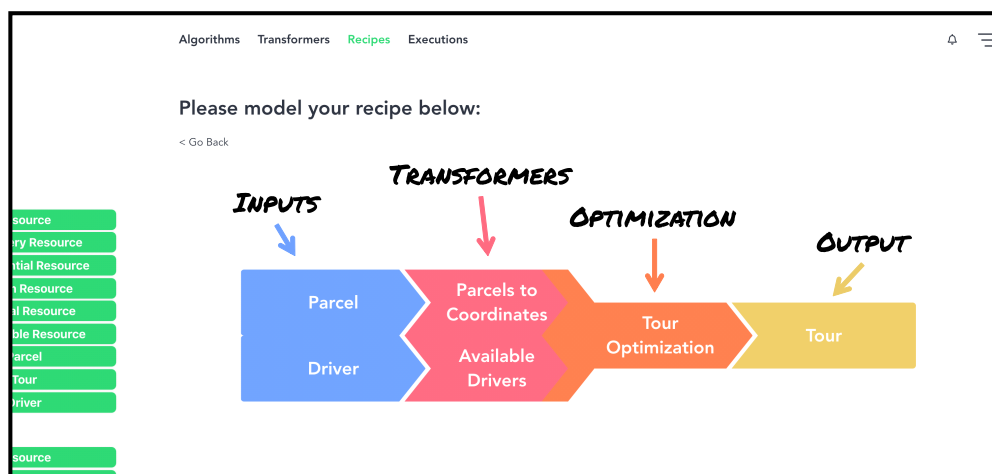


Figure 2: Example of a class hierarchy of shipment resources



**Figure 3:** Annotated screenshot of Rembrandt illustrating the recipe definition for defining an allocation mechanism in the *Resource Optimization* component.

### 2.3. Resource Manager Workflow

Working with the platform follows a clear sequence of steps. To execute a complete resource management cycle, a few steps have to be performed: First, the general structure of resources has to be modelled by creating a type hierarchy. As soon as this is done, concrete resource representations can be instantiated and later modified. In parallel to the instantiation, the resource allocation can be defined. This includes registering the available optimization algorithms, matching them to corresponding resources as inputs and outputs, and creating appropriate recipes. Lastly, resource allocations can be initiated by a process execution engine or a simulator. The concrete step-by-step guide can be found here: <https://rembrandt.gitbook.io/docs/use-case-guide/set-up>.

## 3. Demonstration

As mentioned before, our tool is stand-alone and thus runs separately from any process engine. However, to fully leverage the benefits of our tool, an integration into an existing process execution engine or simulator is necessary. The tool can be integrated into existing approaches by using the documented application programming interface (API) based on REST/HTTP. For demonstration purposes, we have integrated Rembrandt with the process engine Chimera<sup>4</sup>, the process modeler Gryphon<sup>5</sup> and the simulation tool Scylla<sup>6</sup> as shown in the following.

**Integration with a Process Execution Engine** In the first use case, we integrated Rembrandt with the process engine Chimera [5]. Based on the use case of a last mile parcel delivery [6], we observed the need to extend existing BPMS in our past work [3]. We extended

<sup>4</sup><https://bptlab.github.io/chimera>

<sup>5</sup><https://bptlab.github.io/gryphon>

<sup>6</sup><https://bptlab.github.io/scylla>

the BPMS in two ways: First, we extended the process modeler, so that resource type information and goals for the optimization can be configured in the model for each task if necessary. Second, we extended the execution engine so that a request is sent via an extended Service Task that calls the REST interface of Rembrandt and triggers the resource allocation<sup>7</sup>.

**Integration with a Process Simulator** We integrated the functionalities of Rembrandt<sup>8</sup> into an extensible process simulator, called Scylla [7], to demonstrate that it can enhance process analysis capabilities of simulators. Traditionally, simulators only provide simplistic resource allocations. We extended this behaviour<sup>9</sup>, so that more sophisticated algorithms can be used, therefore allowing for example to identify the impact of different resource allocations in a process.

## Acknowledgments

The research leading to these results has been partly funded by the BMWi under grant agreement 01MD18012C, Project SMile.  
<http://smile-project.de>

## References

- [1] C. Cabanillas, Process-and resource-aware information systems, in: EDOC, 2016 IEEE 20th International, IEEE, 2016, pp. 1–10.
- [2] G. Havur, C. Cabanillas, J. Mendling, A. Polleres, Resource allocation with dependencies in business process management systems, in: BPM, Springer, 2016, pp. 3–19.
- [3] S. Ihde, L. Pufahl, M.-B. Lin, A. Goel, M. Weske, Optimized resource allocations in business process models, in: Business Process Management Forum. BPM 2019., Springer, 2019, pp. 55–71.
- [4] J. Munkres, Algorithms for the assignment and transportation problems, *Journal of the society for industrial and applied mathematics* 5 (1957) 32–38.
- [5] S. Haarmann, N. Podlesny, M. Hewelt, A. Meyer, M. Weske, Production case management: A prototypical process engine to execute flexible business processes., in: BPM (Demos), 2015, pp. 110–114.
- [6] L. Pufahl, S. Ihde, M. Glöckner, B. Franczyk, B. Paulus, M. Weske, Countering congestion: A white-label platform for the last mile parcel delivery, in: 23rd International Conference on Business Information Systems, Springer, 2020, pp. 210–223.
- [7] L. Pufahl, T. Y. Wong, M. Weske, Design of an extensible bpmn process simulator, in: International conference on business process management, Springer, 2017, pp. 782–795.

---

<sup>7</sup><https://rembrandt.gitbook.io/docs/user-guide-1/resource-allocation>

<sup>8</sup><https://github.com/bptlab/rembrandt-backend/tree/feature/BPST-Analysis>

<sup>9</sup><https://github.com/bptlab/scylla/tree/rembrandt>