ARMS: Activity-Resource Modelling Simulator

ARMS: Activity-Resource Modelling Simulator

Angelo Di Iorio[1 [0000−0002−6893−7452]], Marco Ferrati[1 [0009−0001−3225−3710]], and Davide Rossi[1 [0000−0003−2269−8094]]

Department of Computer Science and Engineering
University of Bologna, Bologna, Italy
{angelo.diiorio,marco.ferrati2,daviderossi}@unibo.it

Abstract. Activity-based process modeling languages, such as Business Process Model and Notation (BPMN), face challenges in accurately representing manufacturing processes, limiting their effectiveness for simulation purposes. In contrast, specialized manufacturing process simulation tools typically adopt a resource-based approach which, while effective in certain aspects, often lacks robust process modeling capabilities and requires custom coding to define detailed system behaviors. Although prior research has explored extending BPMN to better support manufacturing scenarios, there is still a lack of tools that integrate both activity-centric and resource-centric approaches for comprehensive modeling and simulation.
In this work, we address this gap by proposing a novel metamodel that extends BPMN with enhanced resource capabilities. We present a process modeler that allows users to design manufacturing processes and a corresponding simulator that uses these models to generate insightful information about the process performance through simulation.
The solution proposed is then validated using both made-up and real-world manufacturing scenarios.

Keywords: Manufacturing · simulation · modeling · BPMN

1 Introduction

Most available simulation proposals in the manufacturing domain, including academic prototypes and commercial products, are based on the resource-centric approach (such as FlexSim®, AnyLogic®, Arena®, Warteschlangensimulator [8]). The underlying engine for all these products uses a Discrete Event Simulation (DES) approach [7][6], where the production floor is modeled as a queuing system. In these systems, each *executor* is represented by an event queue, and routes are used to link executors, carrying parts or products. These solutions have proven capable of creating sophisticated, detailed simulations that accurately describe the dynamic evolution of the production processes. However, their use requires specific skills: they adopt non-standard modeling notations, often comprising a visual representation enriched with ad-hoc code (written in a variety of general-purpose or domain-specific languages) to describe the production process in sufficient detail. In this context, by code we do not mean

simple expressions, which are almost unavoidable when modeling the behavior of a complex system (e.g., those used to implement branching decisions), but rather algorithms that influence the overall semantics of the process. Although the use of this kind of code is usually presented as reserved for intricate scenarios, it often becomes necessary to represent the complexity of most real-world manufacturing processes, which hampers the adoption of these tools by users with limited programming proficiency.

Past research has attempted to address this problem by proposing modeling solutions capable of capturing the structure of processes, even highly complex ones, without relying on coding. Most of these proposals adopt BPMN (Business Process Model and Notation) [12], the de facto standard for modeling, enacting, and simulating business processes, as a starting point, enriching it with elements suited to the specific requirements of the manufacturing domain. The core element of a BPMN model is the action, an atomic unit of work that must be executed for the process to progress. This marks a fundamental distinction from the resource-centric approaches previously discussed. To highlight this difference, we refer to the BPMN approach as an activity-centric one.

The adoption of BPMN is justified by its ability to describe complex processes without the need to resort to coding, as demonstrated by the variety of workflow patterns it supports [15]. The designers of BPMN also sought to define a language in which nearly all semantically relevant elements have a notational counterpart, ensuring that the overall semantics of the model are clearly represented in a corresponding diagram. However, this does not imply that BPMN is "superior" to other approaches; its ability to express rich semantics "just by the diagram" is due to its limited support for concerns outside the control-flow perspective [9]. As a result, the use of BPMN for modeling production processes is hindered by its limited capacity to model resources and the relationships between resources and activities.

More specifically, BPMN's metamodel includes *Resource* and *ResourceRole* classes, but their specification is deliberately vague (e.g., "[...] can be Human Resources as well as any other resource assigned to Activities during Process execution time" [12]). Resources can be associated with processes and activities, but the only semantically defined relationship between an activity and a resource role is the one in which the resource role appears as a *Performer*, i.e., the executor to which the activity is assigned.

The aim of our work is to enable analysts to create simulation models of manufacturing processes that capture as much semantics as possible from their graphical representation and from the element's properties (all of which can be edited in a modeling tool), eliminating the necessity to resort to coding for most real-world processes.

With this objective in mind, this paper presents the development of a simulator and a modeler that allow the simulation of manufacturing scenarios, modeled using a mix of activity-based and resource-based approaches, and simulated with a discrete event simulator capable of parsing those models.

The contributions of this paper are:

- an extension of BPMN able to capture most manufacturing processes with a simple and coherent notation;
- a modeler capable of modeling processes that relates activities with resource related concepts;
- a simulator which accepts the model produced by the modeler and simulate it providing insightful data regarding the process described by the model.

The rest of the paper is structured as follows. Section 2 reviews the related works, providing an overview of existing research on BPMN extension and resource-based simulation approaches in manufacturing. In Section 3, we describe our methodology, outlining the proposed metamodel, and the development of the process modeler and the simulator. Section 4 presents the evaluation, where we assess the effectiveness of our approach through a real world case study. Finally, section 5 discusses the conclusions and future works, summarizing our results and highlighting potential directions for further research and development.

2 Related works

The field of process modeling and simulation is vast and well-established. Various activity-centric process modelling languages and techniques are available and used in practice [19], both BPMN-based and non-BPMN-based. Notable examples based on BPMN include Scylla and BPSim. Scylla, based on the work of Pufahl, Wong, and Weske, is an extensible BPMN process simulator that supports the simulation of complex scenarios with a plugin-based architecture [14]. We considered adopting Scylla, but we realized that the plugins we needed to develop would have been too convoluted, resulting in a very complex architecture. BPSim [20], on the other hand, is a WfMC (Workflow Management Coalition) standard that integrates with BPMN to define scenarios and simulation parameters. However, it lacks the capability to model rich resource relationships, which are commonly needed in manufacturing scenarios. In manufacturing process modeling and simulation, BPMN has been employed in several studies, with extensions made to better capture the nuances of manufacturing workflows [3][21][5][13]. Despite the endeavour, they concentrate only on few aspects of the manufacturing process. Various non-BPMN-based alternatives have been considered as well. For instance, [4] compares six notations (flowchart, Petri net, DFD, Role activity diagram, BPMN, and Business use case) concluding that only Petri nets and BPMN are suited for simulation. DPMN [17] attempts to integrate BPMN with Discrete Event Simulation (DES) and in [18] the work is extended to considering resource-constrained activities. Although it focuses on the direction of mixing activity-centric with resource-centric modeling it lacks some relevant features of a manufacturing environment such as many-to-many relationships between activities and performers and the possibility to specify how products transforms during the process. As an alternative to activity-centric modeling, discrete event simulation (DES) has been often proposed for simulating manufacturing processes [2][11][10][1]. Although DES provides robust simulation capabilities, modeling manufacturing processes using this method can

be challenging, as it typically requires specialized skills, such as learning a non-standard notation or relying on programming to model more complex scenarios.

The novelty of our proposal lies in its integration of a resource-centric approach within an activity-centric metamodel, effectively representing individual executors and their compatibility with specific activities. Furthermore, the metamodel accounts for product transformations throughout the process, as well as the executors' accessory requirements. Our solution provides users with both a modeler and a simulator, enabling the modeling of complex resource allocation and transformation using a BPMN extension. This approach eliminates the need for ad-hoc coding and significantly reduces the skill requirements compared to traditional resource-centric simulators.

3 Methodology

Our approach consists in complementing an existing activity-based approach which primarily addresses the control-flow perspective, namely BPMN, with additional concepts meant to capture a resource perspective able to describe elements that are pertinent to the simulation of manufacturing processes. Given our aim of conducting simulations, we designed a metamodel that allows users to express detailed information necessary for such analysis. To achieve these objectives, we have developed two complementary software components: a modeler and a simulation engine. As illustrated in figure 1, users interact with these components to model and simulate processes. The modeler enables users to define the process, including properties necessary for simulation. Once the process is modeled, users can generate a file that represents the designed process, which serves as input to the simulation engine. The simulator processes this input and executes the simulation. Upon completion, users can review the simulation results and derive insights from the collected data. For example they could identify bottlenecks or visualize heat-maps on the model representation.

3.1 Metamodel

The metamodel was designed from two interwoven activities: literature analysis and domain expert elicitation. Specifically, we interacted with two domain experts from a consulting firm with interviews and with sessions of participatory design [16] in which we co-modeled scenarios they proposed with BMPN, annotating all relevant concepts not addressed by the notation.

The metamodel described in this section, illustrated in figure 2, comprises two categories of elements: structural and dynamic, represented in the diagram with distinct colors. Structural elements (depicted in blue) define the process' elements and parameters, whereas dynamic elements (depicted in green) are runtime entities that are instantiated and altered during the simulation.

The primary concepts are: *Activity*, *Executor*, *Product*, and *Accessory*. *Activity* represents a task within a manufacturing process; it serves as a natural starting point given the activity-centric approach we embraced. In the BPMN

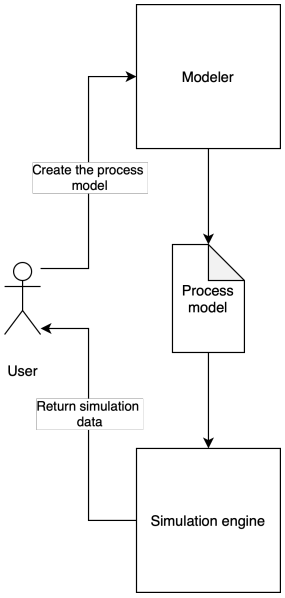


Fig. 1: System-User interaction workflow

metamodel it can be aligned with the corresponding *Activity* entity. Activities can be grouped together and executed under some conditions (for instance, using a washing machine that can wash 15 items at the same time). Prioritization strategies are also necessary since different activities might be competing for using the same resources. The attribute *priority* was indeed introduced to address this last issues.

The *Executor* entity represents any kind of resource engaged in performing an activity. They can represent humans or production machines. *Executors* are characterized by the attribute *quantity* which specifies their number of available executors with the same characteristics. When adopting the metamodel with BPMN this entity, as explained in Section 1, can be aligned with *Performer*.

A *Product* is any object that undergoes processing and transformation during the manufacturing process, representing a single item moving throughout the production floor. Its life-cycle will be managed by the simulator. It is a dynamic entity of the metamodel. Therefore, it is expected to be created during the simulation.

To enable users to model the different types of products present in the manufacturing process, we introduced the *ProductFamily* entity.

Accessory represents a resource that might be necessary for an executor to effectively execute a task. These are passive elements that contribute to the process. The *quantity* attribute captures scenarios where there's a limited number of these resources available to executors. For example, consider a worker tasked

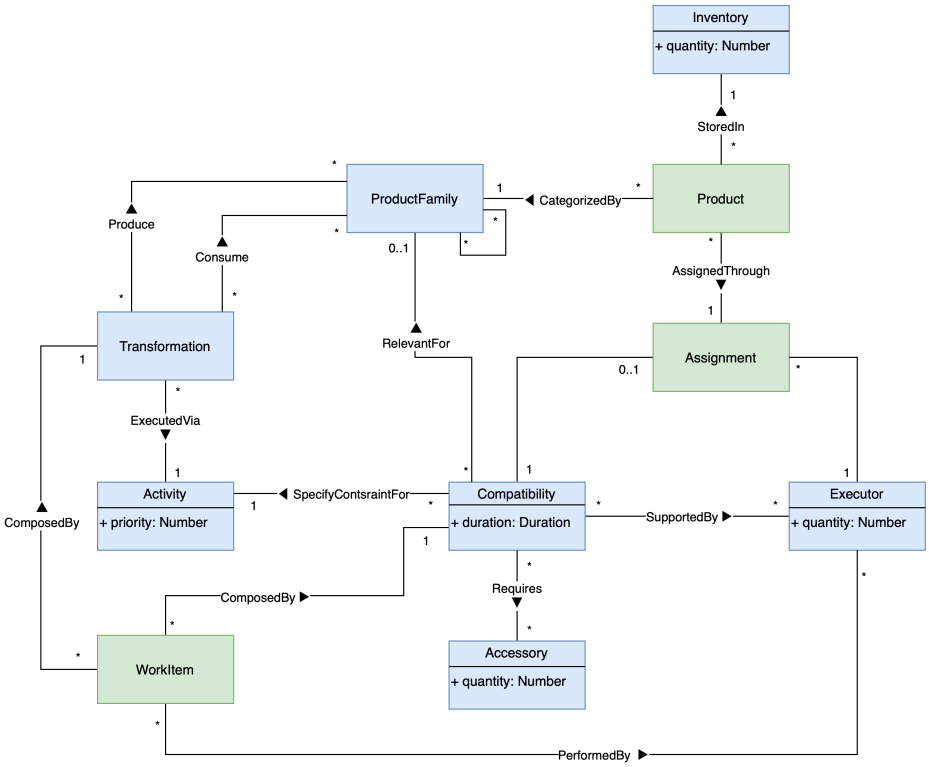


Fig. 2: Metamodel diagram

with “Digging a hole” who requires an auxiliary resource: a shovel. The shovel is an accessory and there could be only two shovels for three workers.

A *WorkItem* is another dynamic element and it is generated upon the activation of a task, with one or more potential *Executor*s identified based on compatibility criteria to carry it out. When specific *Executor*s are assigned to these *WorkItem*s, an *Assignment* instance is created. This assignment process can follow various policies, and it represents a well-known optimization problem. The simplest approach is an eager policy, where unassigned executors continuously search for available compatible *WorkItem*s, claim them for execution, and mark them as unavailable to other compatible executors.

Finally, *Inventory* represents the quantity of available products from which an activity can draw upon. It can also model all those situations where products are stocked somewhere during the production process.

After the definition of the fundamental entities let us discuss the associations between these.

Compatibility relates *Activity*, *ProductFamily*, and *Executor*. It is very common, in fact, that a given product can only be produced by a given machine. Similarly, some operators are specialized for executing some tasks. *Compatibility* allows users to describe which executor can work on which kind of product in an activity. Moreover, it carries the information about how long the executor takes to perform the activity. Very often, in fact, the same task can be executed by different executors, but each of them takes a different time to complete that task. For instance, a modern machine could be more efficient than an older one, or their performance can be affected by external factors.

From the perspective of an activity, *Compatibility* can be thought of as a matrix. For example, tables 1a and 1b show an example of two compatibility matrices for activities “3D-printing handle” and “3D-printing box”. Each activity has columns representing the potential executors that can be assigned to that activity: Printer1 (P1), Printer2 (P2), and Printer3 (P3) for “3D-printing handle” and Printer1 (P1), Printer4 (P4), and Printer5 (P5) for “3D-printing box”. In the rows we have the products that will be worked on that activities: “PLA” and “ABS”. The intersection indicates which executor can work on which product for the activity: in “3D-printing handle”, “PLA” can be worked by P1 and P2, and “ABS” can be worked by P2 and P3. In “3D-printing box”, “PLA” can be worked by P1 and P5, while “ABS” can be worked by P1 and P4.

	P1	P2	P3
PLA	•	•	
ABS		•	•

(a) Compatibility matrix for activity “3D-printing handle”

	P1	P4	P5
PLA	•		•
ABS	•	•	

(b) Compatibility matrix for activity “3D-printing box”

Table 1: Compatibility matrices

Note also that a given activity might be performed by an executor by using one or more resources together — for instance, an operator using a washing machine and some detergents — or by different executors that work together — an assembly line with multiple operators. The association between *Compatibility* and *Accessory* captures these relations and their cardinality.

Assignment represents the history of assignments, detailing which activities and specific products each executor has worked on. For example, take the scenario described in table 1a, the simulator can decide to assign the executor P1 to work on a product belonging to the PLA family for the activity "3D-printing handle". This decision is allowed because the compatibility between activity, executor, and product family is defined through the Compatibility entity.

It is quite common that the same executor is selected to perform different activities on the same resource in order to increase efficiency. Consider for instance some quality check activities on some products being produced: it would be better if an executor checks the same product. This is only possible when tracing the associations between activities, products and executors.

To explain how products are transformed during the process, there are two associations. There is a self-association in *ProductFamily*, and an association class (*Transformation*) between *Activity* and *ProductFamily*. The self-association in *ProductFamily* allows us to describe from which products a product comes and into which products it can transform. In addition, the auxiliary entity *Transformation* indicates in which activity a product undergoes a transformation. Since many transformations can be possible in an activity, we used many-to-many cardinality on *Consume*/*Produce*. Moreover, this entity allows us to model a batch activity. Through the *Consume* and *Produce* relationships we can model scenarios where to perform an activity we have to wait for a certain number of items to be ready.

Finally, the association *Inventory-Product* serves to designate the specific inventory where a particular product can be located and accessed. This association is between *Inventory* and *Product*, and not between *Inventory* and *ProductFamily* because we want to keep track of the specific products inside the inventory and not only the family of these products.

3.2 Model editing

Ideally, extending BPMN with the metamodel we just discussed also entails determining notational symbols for each element of our metamodel, in line with the principle of capturing "as much semantics as possible from the diagram," which characterizes BPMN. However, this would have resulted in very convoluted diagrams, severely hampering readability and understandability. To address this, we adopted a pragmatic approach, opting to represent *Executors* notationally (using hexagon shapes), to partially depict the *Compatibility* relationship using dashed lines connecting executors and activities, and the *Inventory* entities as data storage. All other elements and relationships are rendered with specific input widget forms, tailored to each element.

3.3 Tool

Modeller The modeler is designed to extend standard BPMN to include the resource elements described before. This extension enables users to capture resource interaction within business process diagrams, ensuring that both process and their associated resources are represented. The modeler[1], shown in figure 3, has been implemented as a web application using two core technologies: React[2] and BPMN-js[3]. By leveraging BPMN-js, which provides a modeler for standard BPMN, we were able to focus on extending its capabilities to support the aforementioned resource modeling features. Specifically, our solution implements the following features:

- a BPMN extension that incorporates new elements such as: executors, new connections, inventories, compatibilities and transformations;
- a rendering mechanism to display executors and the connection between executors and activities;
- a linter enforcing validation rules for the new elements. For instance, the modeler shows errors when an executor or activity is not properly connected to other elements or when they are missing information necessary for the simulation;
- specific widgets to specify compatibilities between executors, activities, and product properties, as well as transformations from one product to another, prioritization of activities, assignment of accessories to executors, and requests for specific products amount to be produced.

The modeler allows users to download a `.bpmn` file that represents the process they want to simulate. This file follows the standard BPMN format but includes additional elements. Specifically, within the `<bpmn:extensionElements>` tag, four new elements are introduced:

- *Compatibility*: represents a compatibility between an activity and an executor. It specifies the product properties and the duration;
- *Transformation*: it is associated with an activity and specified the product properties is applies to, the updated product properties after the activity completion, the required input products and the resulting output products;
- *ProductRequest*: represent which products we want to produce during the process;
- *Accessory*: represents which and how many accessories are available in the process.

Moreover, *Executor* elements has been added to the list of elements among *Task*, *Gateway*, and *Connection*.

The following example represents the transformation in an assembling activity. Three components (“Frontal frame A”, “Right stem A”, and “Left stem A”)

[1] https://github.com/jjocram/ARMS-Editor
[2] https://react.dev
[3] https://bpmn.io/toolkit/bpmn-js/

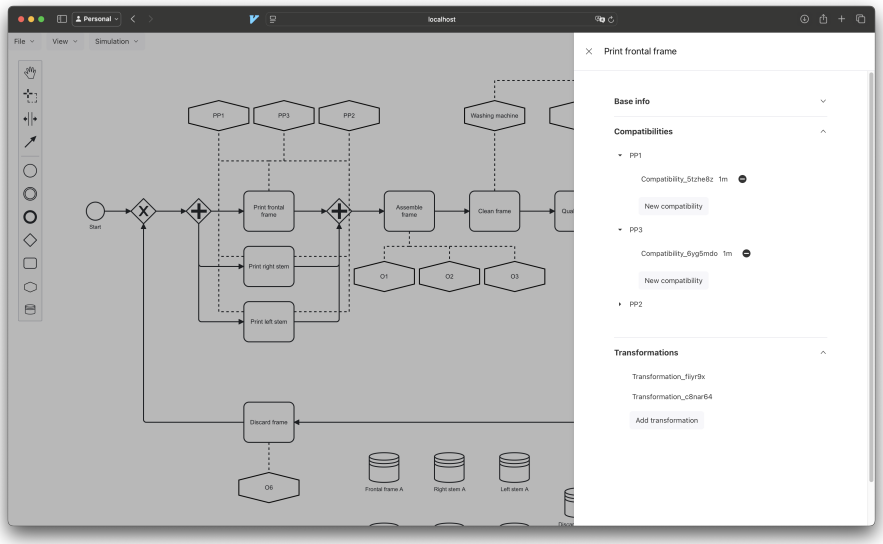


Fig. 3: Screenshot of the modeler

from three inventories are taken as input and consumed to produce an "Assembled frame A". The quantity attribute in the *TransformationInputs* represents how many items are necessary. Figure 4 represents the widget to model this transformation.

```
<factory:transformation id="Transformation_26c94ww" activityId="
    Activity_143dkap">
  <factory:transformationProductProperty
    key="type"
    value="A" />
  <factory:transformationInput
    id="Inventory_0wqk0gz"
    quantity="1" />
  <factory:transformationInput
    id="Inventory_0cd5lgu"
    quantity="1" />
  <factory:transformationInput
    id="Inventory_0e3a21a"
    quantity="1" />
</factory:transformation>
```

Regarding how compatibilities are saved, each of them represent a connection between an activity and an executor. Moreover, they specify which product's properties are required and which accessories are needed. The compatibility matrices showed in tables 1a is represented as follow:

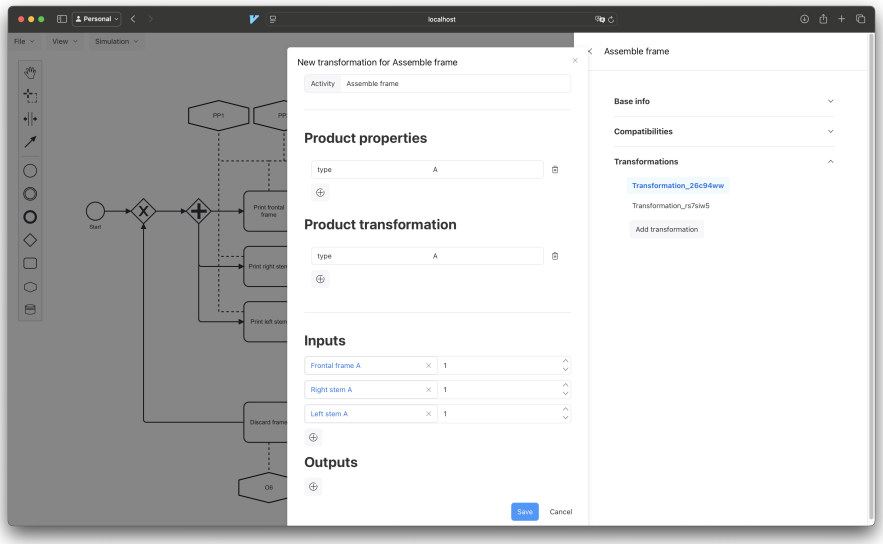


Fig. 4: Widget for modeling transformations

```
<factory:compatibility id="Compatibility_50lyth8" time="1"
    timeUnit="m" idActivity="3D-printing handle" idExecutor="P1">
    <factory:productProperties key="type" value="PLA"/>
</factory:compatibility>

<factory:compatibility id="Compatibility_50lytr7" time="1"
    timeUnit="m" idActivity="3D-printing handle" idExecutor="P2">
    <factory:productProperties key="type" value="PLA"/>
</factory:compatibility>

<factory:compatibility id="Compatibility_50lyt5h" time="1"
    timeUnit="m" idActivity="3D-printing handle" idExecutor="P2">
    <factory:productProperties key="type" value="ABS"/>
</factory:compatibility>

<factory:compatibility id="Compatibility_50lye9h" time="1"
    timeUnit="m" idActivity="3D-printing handle" idExecutor="P3">
    <factory:productProperties key="type" value="ABS"/>
</factory:compatibility>
```

Figure 5 shows the widget to model the first compatibility.

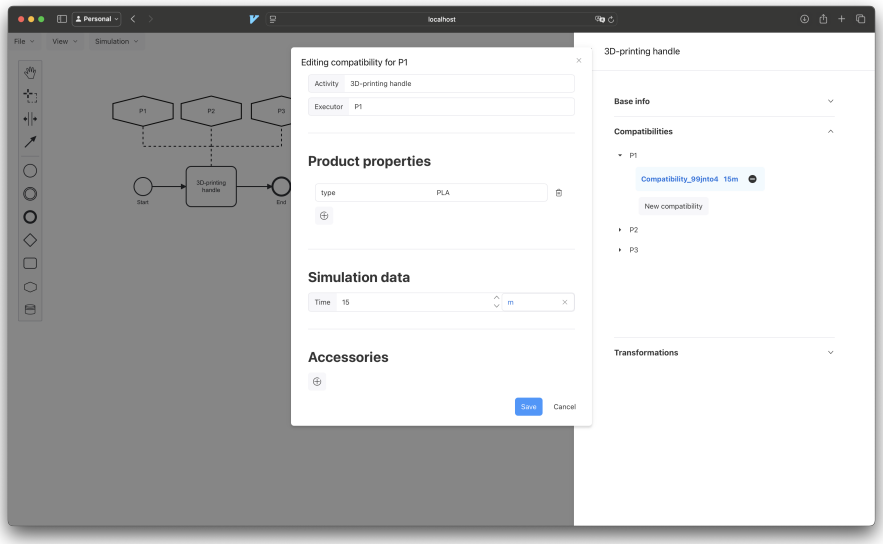


Fig. 5: Widget for modeling compatibilities

Simulation Engine The simulation engine[4] is a discrete-event simulator built on top of Kalasim[5]. It operates based on a Petri net-inspired formalism, incorporating places and transitions. In this implementation, places hold a list of *ControlToken*s, which are moved by transitions when specific conditions are met. Each BPMN element is implemented as a class, using places and transitions to match its execution semantics. Alongside *ControlToken*s, additional places store *ProductToken*s which represent the products undergoing processing. These tokens enable tracking of product modifications throughout the process, allowing compatibility checks based on product properties rather than type alone. Furthermore, *Accessories* are also implemented as places, with the number of tokens corresponding to the quantities specified in the model. This structure supports flexible and accurate representation of dynamic processes and product variations within the simulation environment.

There are a couple of other entities which facilitate accurate representation of the model:

- *Compatibility*: this entity, embedded within an *Activity*, references an *Executor* and specifies product properties to guide selection. It also includes a list of required accessories and the task duration;
- *Transformation*: representing the *Transformation* entity in the metamodel diagram, this entity contains information on how to apply a transformation

[4] https://github.com/jjocram/ARMS-Simulation-Engine
[5] https://www.kalasim.org

to a *ProductToken*, as well as the required input resources and the resulting outputs allowing to model the batch size.

The simulation begins at the BPMN *StartEvent*, where the initial number of *ControlTokens* and *ProductTokens* is generated according to the *ProductRequest*s. Each token is then distributed to the subsequent elements in the process. Independently, each element checks if its transition conditions are met. If so, it executes its specific logic and this cycle is repeated until all tokens reach an *EndEvent*.

Activities and executors operate with additional logic. Each activity maintains a list of compatibilities, defining which executors can process a product based on its properties. When *ControlToken* and *ProductToken* arrive at an activity, the appropriate executor is selected, and a new *WorkItem* is added to that executor's queue. A *WorkItem* represents a specific task that an executor must complete. It includes details such as the required duration, the necessary accessories, and the depletable resources needed for execution. Additionally, it specifies which resources will be produced as a result of completing the task.

Each *Executor* monitors its queue for pending work items. Upon detecting a work item, the executor retrieves the necessary tokens from the accessory places, consumes the tokens specified in the transformation's input, advances the simulation time according to the work item's duration, generates tokens for the transformation's output, and finally moves both the *ControlToken* and *ProductToken* forward in the process.

The simulation ends when no further events remain in the simulation queue, indicating that all *ControlToken*s have reached an *EndEvent*. At this stage, the simulator saves a set of simulation data, including resource utilization levels over time, resource statistics, and component utilization metrics. This data, automatically collected by Kalasim, can then be further analyzed to draw insights and conclusions about the simulated process. For example they could be used to identify bottlenecks or to plot a heat-map on top of the model representation.

4 Evaluation

To evaluate our solution, we asked for help from the domain experts to combine real-world scenarios and create a unique and more complex one. The result is a process which describes how frames for eyeglasses are produced in a manufacturing environment.

We then built a model for this scenario with our modeler, downloaded and given as input to the simulator. The scenario can be summarized as follows:

> Two types of frames for eyeglasses are manufactured on a production line. The goal is to produce 100 frames of type A and 100 frames of type B. Each frame consists of two stems (left and right) and the frontal lens mount. These components are printed on a press printer. There are three press printers in the production area (PP1, PP2, PP3). Type A stems can only be printed on PP1, while type B stems can be printed on PP1 and

PP2. The frontal frame for both types A and B can be printed with any of the three printers. Each press printer takes 1 minute to print a piece and requires the corresponding mould, with only one mould available for each piece. This means that two printers cannot print the same item at the same time. The three printers have a start-up time of 10 minutes, and it is preferable to use a printer that is already running. In addition, the printing of frontal frames has a higher priority because it is a more complex activity. An operator manually assembles the final frame. There are three operators (O1, O2, and O3) who can assemble the frames. O1 takes 1 minute, while O2 and O3 each take 2 minutes each.
Once assembled, the frames are placed in a tumbler, of which there is only one in the production area. The frames are processed in the machine for 24 hours and the machine has a capacity of 15 frames at a time.
A quality check is carried out by two operators (O4, and O5) on one item out of every 20 frames and it takes 2 minutes. If a frame is found to have some defects, it is discarded and has to be reprinted. The frames are then stored in a warehouse, waiting for the next activity.
Every Monday, a lorry arrives at the production site to collect all the frames that are ready for painting. The painting process is done off-site and takes 2 days. Once the frames are back on the production floor, they undergo another round of cleaning, and a final quality check.

Figure 6 represents the process as modeled in the modeler. Keep in mind that not every entity will be showed in the diagram. Some of them are detailed with specific widgets in the modeler when the related diagram elements are selected (as shown in figure 4 and figure 5).

After the modeling phase, we gave the model file as input to the simulation engine which performed the simulation and returned insightful data about the process. For example, we were interested in analyzing the time required to produce the final glass products. To obtain these results, we recorded the accumulation of *ProductTokens* in the end place over time. An example of the diagram showing the growth levels of the two final products is plotted in figure 7.

5 Conclusions and future works

The overall goal of our work is to empower individuals with limited programming skills to design manufacturing process models that can be directly translated into simulation data and used within simulation tools.

Toward this goal, we presented three interlinked components: (i) a BPMN extension to visually edit models representing manufacturing processes, (ii) a graphical modeling tool that integrates it seamlessly and export simulation data and (iii) a simulator capable of processing these data straightforwardly.

We propose to enhance BPMN diagrams with elements derived from a resource-centric approach and simulation data. Such an approach proved to facilitate the simulation analysis of even complex manufacturing processes, limiting the complexity for the final users.

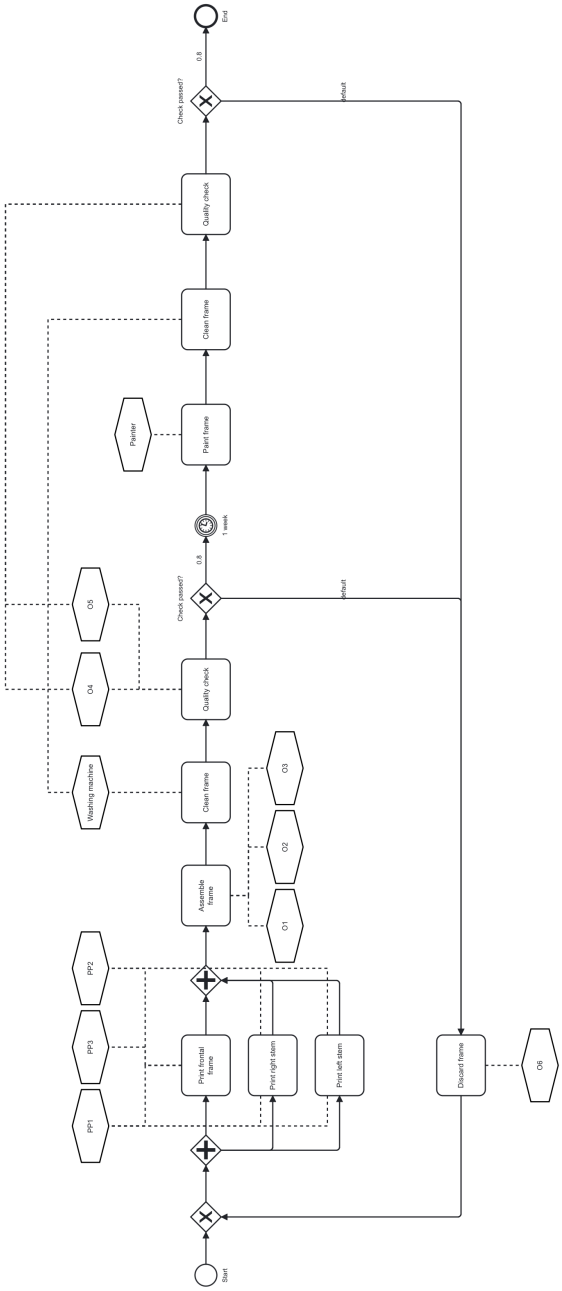
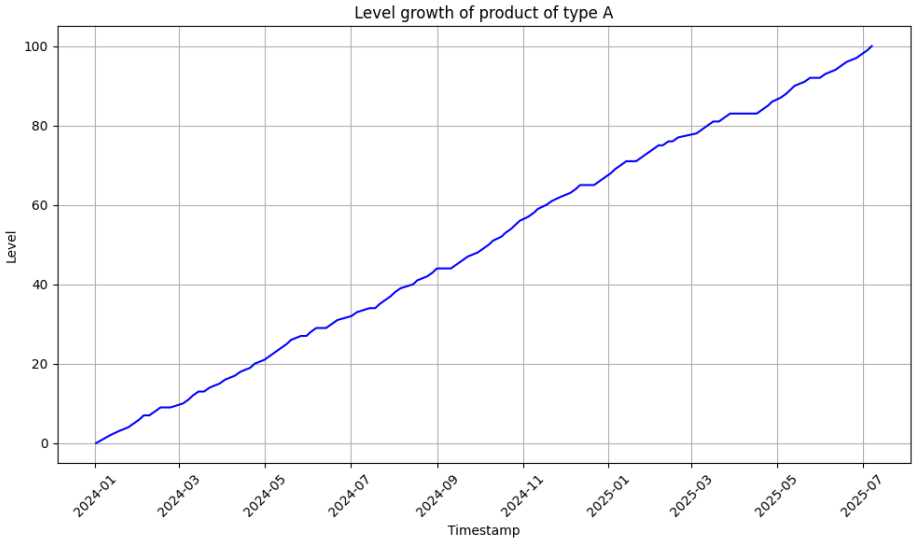
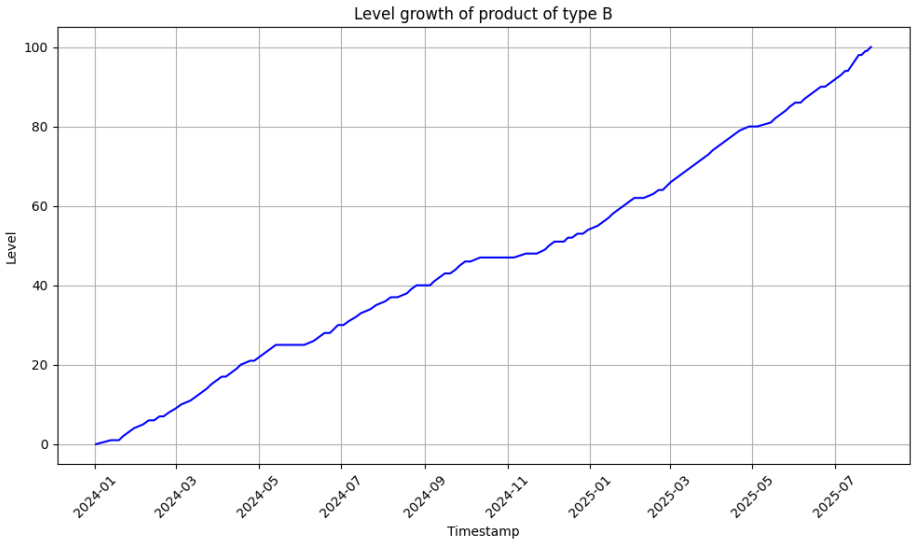


Fig. 6: Diagram representing the scenario's activities, resources, and how they are interconnected



(a) Level growth of type A.



(b) Level growth of type B.

Fig. 7: Level growth of glasses of type A and B

It is worth highlighting the main threats to validity of our work at this stage.

First of all, the metamodel is built on top of a subset of BPMN that, for instance, does not include sub-processes, groups, and pools. This can result in issues with the modeling of large processes. We are actively working to support the full specification and we are confident that this will be possible with no major reconsideration of the current design choices.

The current validation is also limited. It is true that we built the metamodel in collaboration with domain experts and we verified it on real cases, thus ensuring that common issues are addressed and common manufacturing processes are captured but we have only performed limited simulations and we need to test the overall system on a larger scale.

Related to this point, we still have to assess the generalizability of our approach. It might be more difficult to apply it where a higher degree of detail is required, for instance to control the production floor and the interaction between machines and users. Some manufacturing processes could not satisfactorily be addressed by our approach but we expect these cases to be quite limited.

Some more work is also needed on the implementation side. First of all we are building an integrated environment, where users can run the simulation directly within the modeler. More importantly, we plan to include reporting functionalities that show the results of the simulation directly in the original diagram, for instance through heat-maps that would allow users to easily identify bottlenecks and wastes. The inclusion in the metamodel of further BPMN elements, as mentioned above, will also require us to include these in the visual modeler and possibly add panels to configure these elements and their properties.

A further improvement might also be to extend the visual modeler to control simulation parameters. This is actually not trivial, not because of implementation difficulties, but because of the risks of burdening the users' interaction and to overload the metamodel. There is also a technical counterpart to this issue: which is the best place and format to store simulation data? BPMN is well integrated with BPMSim, a standard expressly designed to include simulation parameters in BPMN, thus a very good candidate to be integrated in our system. But BPSim does not fully supports our extension so we would have to extend it as well. Again, it is a trade-off that needs a deeper analysis that we have only started at this stage.

Last but not least, we plan to explore how to support the definition of decision strategies. In the current implementation, some basic policies are hard-coded in BPMN gateways and users have no capabilities for configuring them. We plan to let users define guards based on a simple expression language or to use a set of predefined policies and predicates, but this solution requires further investigation.

Acknowledgement. We would like to express our gratitude to the NextGenerationEU for its co-funding, which made this research possible. We also extend our thanks to Bonfiglioli Consulting, an additional co-founder, for their support and collaboration. Special appreciation goes to Prof. Dr. Mathias Weske for his

constructive ideas and insightful feedback throughout the metamodel definition process. Additionally, we acknowledge Roberto Mitugno for his contributions to the modeler software.

References

1. Agalianos, K., Ponis, S.T., Aretoulaki, E., Plakas, G., Efthymiou, O.: Discrete Event Simulation and Digital Twins: Review and Challenges for Logistics. Procedia Manufacturing **51**, 1636–1641 (Jan 2020). https://doi.org/10.1016/j.promfg.2020.10.228, https://www.sciencedirect.com/science/article/pii/S2351978920320990
2. Babulak, E., Wang, M.: Discrete event simulation: State of the art. Aitor Goti (Hg.): Discrete Event Simulations. Rijeka, Kroatien: Sciyo pp. 1–9 (08 2010)
3. Bocciarelli, P., D'Ambrogio, A., Giglio, A., Paglia, E.: A BPMN extension for modeling cyber-physical-production-systems in the context of industry 4.0. In: Fortino, G., Zhou, M., Lukszo, Z., Vasilakos, A.V., Basile, F., Palau, C.E., Liotta, A., Fanti, M.P., Guerrieri, A., Vinci, A. (eds.) 14th IEEE International Conference on Networking, Sensing and Control, ICNSC 2017, Calabria, Italy, May 16-18, 2017. pp. 599–604. IEEE (2017). https://doi.org/10.1109/ICNSC.2017.8000159, https://doi.org/10.1109/ICNSC.2017.8000159
4. Choudhary, R., Riaz, N.: A business process re-engineering approach to transform business process simulation to bpmn model. PLOS ONE **18**(3), 1–25 (03 2023). https://doi.org/10.1371/journal.pone.0277217, https://doi.org/10.1371/journal.pone.0277217
5. Erasmus, J., Vanderfeesten, I.T.P., Traganos, K., Grefen, P.: Using business process models for the specification of manufacturing operations. Comput. Ind. **123**, 103297 (2020). https://doi.org/10.1016/J.COMPIND.2020.103297, https://doi.org/10.1016/j.compind.2020.103297
6. Fishman, G.S.: Discrete-event simulation: modeling, programming, and analysis, vol. 537. Springer (2001)
7. Goldsman, D., Goldsman, P.: Discrete-event simulation. In: Modeling and Simulation in the Systems Engineering Life Cycle: Core Concepts and Accompanying Lectures, pp. 103–109. Springer (2015)
8. Herzog, A.: Simulation mit dem warteschlangensimulator. Angenommen zur Veröffentlichung (2021)
9. Jablonski, S., Bussler, C.: Workflow management - modeling concepts, architecture and implementation. International Thomson (1996)
10. Mourtzis, D.: Simulation in the design and operation of manufacturing systems: state of the art and new trends. International Journal of Production Research **58**(7), 1927–1949 (2020). https://doi.org/10.1080/00207543.2019.1636321
11. Mourtzis, D., Doukas, M., Bernidaki, D.: Simulation in manufacturing: Review and challenges. Procedia Cirp **25**, 213–229 (2014)
12. OMG: Business process model and notation (bpmn) (2011)
13. Pufahl, L., Weske, M.: Batch activities in process modeling and execution. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) Service-Oriented Computing - 11th International Conference, ICSOC 2013, Berlin, Germany, December 2-5, 2013, Proceedings. Lecture Notes in Computer Science, vol. 8274, pp. 283–297. Springer (2013). https://doi.org/10.1007/978-3-642-45005-1_20, https://doi.org/10.1007/978-3-642-45005-1_20

14. Pufahl, L., Wong, T.Y., Weske, M.: Design of an extensible BPMN process simulator. In: Teniente, E., Weidlich, M. (eds.) Business Process Management Workshops - BPM 2017 International Workshops, Barcelona, Spain, September 10-11, 2017, Revised Papers. Lecture Notes in Business Information Processing, vol. 308, pp. 782–795. Springer (2017). https://doi.org/10.1007/978-3-319-74030-0_62, https://doi.org/10.1007/978-3-319-74030-0_62
15. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M., Edmond, D.: Workflow resource patterns: Identification, representation and tool support. In: Pastor, O., e Cunha, J.F. (eds.) Advanced Information Systems Engineering, 17th International Conference, CAiSE 2005, Porto, Portugal, June 13-17, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3520, pp. 216–232. Springer (2005). https://doi.org/10.1007/11431855_16, https://doi.org/10.1007/11431855_16
16. Spinuzzi, C.: The methodology of participatory design. Technical Communication **52**(2), 163–174 (2005), https://www.ingentaconnect.com/content/stc/tc/2005/00000052/00000002/art00005
17. Wagner, G.: Information and process modeling for simulation–part i. Journal of Simulation Engineering **1**, 1–1 (2018)
18. Wagner, G.: Business process modeling and simulation with dpmn: Resource-constrained activities. In: 2020 Winter Simulation Conference (WSC). pp. 45–59 (2020). https://doi.org/10.1109/WSC48552.2020.9383915
19. Weske, M.: Business Process Management - Concepts, Languages, Architectures, Third Edition. Springer (2019). https://doi.org/10.1007/978-3-662-59432-2, https://doi.org/10.1007/978-3-662-59432-2
20. WfMC: Bpsim (2016)
21. Zor, S., Schumm, D., Leymann, F.: A proposal of bpmn extensions for the manufacturing domain. In: Proceedings of the 44th CIRP International Conference on Manufacturing Systems (2011)