Enhancing Simulation Capabilities in Proter

Michal Baczun

MInf Project (Part 1) Report

Master of Informatics
School of Informatics
University of Edinburgh

2021

Abstract

Discrete event simulation is a common method for analysing business processes for the purposes of predictive analytics, optimisation, and exploring possible "what-if" scenarios. The quality of the simulation depends on the model used to describe the process and its environment. Task priorities and priority-based scheduling are one example of an aspect which is hugely beneficial in creating more realistic and accurate models which is unfortunately often overlooked.

Proter is a unique business process simulator with a focus on prioritised processes which enables the simulation of such models in way that more accurately represents how people schedule tasks. However, having originally been built within the narrow context of a research prototype, Proter lacks a number of features encountered in typical use cases and tools for general purpose business process simulation.

Our main goal is to enhance Proter towards a fully functional, general purpose business process simulator, such that it is comparable with existing, state-of-the-art simulators with the addition of its unique priority-based scheduling features. Through this process, we also introduce a concrete set of criteria to evaluate and compare business process simulators.

This thesis presents a collection of essential criteria identified as part of a survey of BPS literature, and a practical evaluation of Proter against existing BPS tools using a handmade dataset of practical examples that test for each of the essential criteria.

Acknowledgements

I would like to thank my supervisor, Dr Petros Papapanagiotou, for his invaluable guidance and mentorship throughout this project.

I would also like to thank my family for their support and motivation.

Table of Contents

Chapter 1

Introduction

Simulation is a particularly useful tool towards understanding a complex system and examining how it behaves. Simulation can be used to gain insight into an existing or proposed situation, to test what-if hypotheses, to see how a system will behave over a longer period of time, or to test a scenario which is too expensive or too dangerous to try out in reality. This is a very common and powerful application of computing, and it is used in a wide variety of disciplines and fields of study.

Business Process Management (BPM) is a particular discipline in which simulation can be very useful. BPM includes various activities such as modeling, discovery, automation, measurement, and optimisation of operational business processes in support of enterprise goals [34, 20]. In this field, **Business Process Simulation (BPS)** is a particularly useful tool which can enable BPM professionals to analyse and optimise business workflows. BPS can be used to consider many potential scenarios for "What-if" analysis, in support of decision making, to identify bottlenecks in a system, or for prediction, making it a very versatile and useful technique.

A number of BPS tools exist for this purpose, but the functionality and capabilities of these tools varies greatly. In this project we present, develop, and evaluate the functionality of our own business process simulator, named **Proter** [26]. Proter is a very capable and unique tool for discrete event simulation, with novel features that can enable a more realistic simulation of business processes. In our effort to perform a comparative evaluation of these features, it is made obvious that Proter lacks some common features which are seen in many other tools. This thesis is a study of how Proter compares to other simulators and what we can do to improve it.

1.1 Motivation

As mentioned, BPS is a useful technique which is commonly used in BPM in order to gain insight into business processes or trial possible scenarios, especially where this it is infeasible in real life. The quality and accuracy of BPS based analysis is directly affected by the level of abstraction and realism that can be achieved in the simulation model and associated tools.

Most business process simulators do not place enough consideration into priorities. In reality, different activities have a different priority, even if this is sometimes implicit.For instance, in the real world if a very important order is placed at a manufacturing facility then it would likely be assembled as soon as possible, or if a very important customer enters your store you might try to get to them as fast as you can, potentially ahead of others. As a surgeon, if you know that a patient in critical condition will arrive on an ambulance in half an hour, you would not start a new operation that will take 3 hours to complete, but instead, to avoid downtime, you might take a few minutes to check on your other patients. This is what Proter does, instead of a greedy approach where as many tasks are started here-and-now, it puts the higher-priority tasks first and ensures these are not delayed in favour of lower-priority ones.

This prioritised approach of Proter is closer to how processes are handled in the real world, and it gives users of the simulator an additional way of expressing how their processes work and how they need to be simulated. This extra degree of control can be very advantageous, and we would like for Proter to be accessible for researchers and BPM professionals so that they can utilise priorities effectively in their simulations.

There are a few obstacles which might discourage interested parties from using Proter, and removing these obstacles is the focus of this thesis. As we discover, there are a number of features which are commonplace among other simulators which were missing in Proter, some of which turn out to be very essential for BPS. Furthermore, the majority of BPS tools use the **Business Process Modelling Notation (BPMN)** as the language to describe the workflow models that they simulate. This is a common notation used across the BPM discipline and is sometimes seen as preferable to alternatives due to its familiarity and widespread usage. This is why in this project we set out to add some essential functionality to Proter and to extend it to be compatible with BPMN models.

1.2 Methodology

To our knowledge there are no standardised criteria or functionality that should be provided by a business process simulator. Because of this, the first course of action is to identify what features are considered to be the most important in the literature, and how Proter stacks up against these expectations especially when compared to other simulators. This compiled set of criteria also serves as a useful output of this thesis and could be used in future work for comparing and evaluating BPS tools.

From there, the most indispensable of these criteria are implemented in Proter. We also implement support for using BPMN models in Proter due to their widespread usage in BPM and in other BPS tools.

Once these features are implemented we are able to compare Proter with other simulators using the exact same BPMN models for simulation. The new simulator functionality which we add is essential in enabling Proter to perform the same simulation scenarios as other tools. The evaluation of Proter asserts that it can replicate the same results as existing tools and allows us to discuss the interesting differences in the tools' functionality and output, with a particular focus on Proter's unique features.

1.3 Project Goals

In summary, this project set out to achieve the following goals:

1. Identify a set of the most important simulation criteria based on tool surveys and other BPS literature (Chapter 4).
2. Implement missing essential simulation capabilities into Proter (Chapter 5).
3. Implement BPMN compatibility, a standard feature of many simulators which also enables us to compare Proter to existing tools using the same BPMN models (Chapter 6).
4. Evaluate Proter's capabilities by comparing it with existing popular BPS tools on the basis of the identified simulation criteria (Chapter 7).
5. Extend the list of proposed essential simulation criteria based on the novel features regarding prioritised processes (Chapter 7).

We begin by introducing some of the necessary background and context of this project in the next chapter. This is followed by a detailed discussion of how we approached each of our goals in their respective chapters.

Chapter 2

Background

In this chapter we discuss the context of simulation within BPM and introduce the modeling notations used in this field before talking briefly about some of the existing BPS tools.

2.1 Business Process Modelling and Simulation

Van der Aalst is one of the most influential authorities in the fields of business process management, with a long history or highly recognised publications in the field. His "Business Process Simulation Survival Guide" [30] stands out as one of the most useful and comprehensive reviews of BPS and relevant concepts and techniques.

According to the Survival Guide, the possible motivations for using a simulation are:

- Gaining insight in an existing or proposed future situation.
- Running a real experiment might be too expensive.
- Real experiments may be too dangerous or unrepeatable.

Business Process Management (BPM) is a discipline which includes various activities such as modeling, discovery, automation, measurement, and optimisation of operational business processes in support of enterprise goals [34, 20]. There exists a number of BPM tools and software to support such activities, but BPM is fundamentally a set of methodologies for managing and transforming business operations while the software is considered auxiliary [9]. Nevertheless, **Business Process Simulation (BPS)** has emerged as one of the most established analysis techniques in BPM [30].

BPS refers to discrete event simulation of business processes within the field of BPM. In BPS we are interested in simulating workflows which consist of tasks that take some time to complete and potentially require the use of certain resources. The advantage of BPS over alternatives such as Markov Chains is that it tends to be flexible, easy to understand, and capable of answering a wide variety of questions. It can be used for a variety of purposes including determining key performance indicators such as finding

average queuing times or resource utilization, trialling possible "what-if" scenarios, predictive analytics, and decision support [30].

Through the practice of BPM we might have a model, or workflow, which describes a business process. A BPM workflow model, typically defined using some modelling notation, describes the appropriate sequence of tasks and decisions for a particular case. Task dependencies and sequences can vary a lot, but there exist a number of typical workflow patterns [32] that can be used to evaluate the expressiveness of different modelling languages.

If we wanted to use BPS to learn more about this workflow, we would need to also describe the environment it operates in, and the relevant aspects of its domain. Each task in the workflow may require specific resources (for example equipment, machinery, specialist workers) and take some amount of time to complete. The duration of the task might be constant, or it could be drawn from a probability distribution. Resources should have certain roles, capacities, and a schedule [27], and tasks may require multiple resources to run.

The environment may consist of different types of resources with different capacities. When a workflow is simulated, we go through each of its tasks, assign the appropriate resources from the environment and execute it in virtual, discrete time. As we execute multiple interleaving workflow cases with different tasks and resource requirements, we start observing the delays, waiting times and bottlenecks caused by the limited available resources. A scheduling strategy becomes essential for deciding which tasks can begin at any given moment. For example, most simulators use a basic first come first served principle for assigning tasks to resources, and if a certain required resource is currently busy then the task cannot begin and it is saved until later. This aspect of BPS research can lead to subjects such as queuing theory and job shop scheduling.

A good BPS study should involve the following [30, 14]:

1. **Arrival process** - This is the process which dictates how new instances arrive in the simulation. For example if a workflow describes the journey of how a product for a specific order is manufactured, then the arrival process will describe the frequency with which new orders arrive at the facility, and the simulation will include a constant flow of old orders being completed and new orders being started. This is how to queuing, bottlenecks, and competition for resources might arise in the system.

2. **Warm-up period** - This is typically a designated period of time at the beginning of a simulation which is ignored in the final results. This is in anticipation of the way that resource utilisation, queuing, or the number of instances currently being handled might take some time to reach their true level when we start simulating from an empty system.

3. **Replications** - Also known as sub-runs, this refers to either running a simulation multiple times or splitting a simulation into multiple segments, and is necessary in obtaining confidence intervals.

4. **Confidence intervals** - These should be calculated for any desired Key Per-

formance Indicator by using the results of many replications and through the application of the central limit theorem.

The effectiveness of BPS depends on the granularity of the model. It is impossible to model the entire domain and choosing the appropriate level of abstraction can be hard: Should the worker operating the machine be modeled as a resource? Should we model that they work slower right before lunch? If their mood affects their productivity, maybe we should model the factors that affect their mood, like weather and personal life? Inadequate modelling of human actors as resources is one of the most common pitfalls in simulation, according to the Survival Guide. Other typical pitfalls it identifies include using the wrong level of detail which can lead to over-complicating of the model, forcing the model to fit by blindly tweaking parameters (instead of finding the reason why the model is not quite right), and omitting replications which leads to incorrect assumptions about confidence intervals for key performance indicators. This is not helped by the fact that many commonly used BPM simulation tools lack certain capabilities, like warm-up periods, multiple resource roles, and replications [24]

2.2 Modeling Notations

Business process modeling notations are integral to BPM. With a modeling notation one can visually represent a workflow, which is useful not only for simulation but also for describing and communicating business processes to other people. Many types of notation exist, but the two most commonly used ones in the business process simulation are *BPMN* and *Petri Nets*.

Petri nets [17], can be used to model and analyse all kinds of processes. In BPS they are the most common alternative to BPMN, featuring in popular simulators such as CPN Tools [12]. In this thesis we focus on BPMN over petri nets due to the popularity and widespread usage in BPM and in the vast majority of popular business process simulators.

2.2.1 BPMN

BPMN (Business Process Modeling Notation) [18] was created with the primary goal of providing a notation which is understandable by both business and technical stakeholders. BPMN defines Business Process Diagrams (BPD) which are based on a flowcharting technique. A basic BPMN diagram can be seen in Figure 2.1.

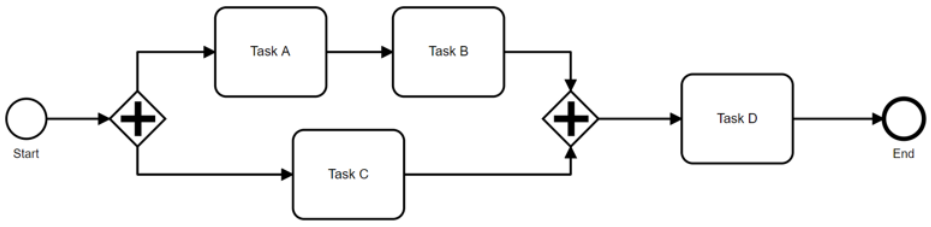


Figure 2.1: A basic BPMN model.

The basic elements of a BPMN diagram are:

- **Flow objects**, of which there are three types:
  - **Events**, denoted by a circle, represent the start and end events in a process, and external variables or stimuli along a path.
  - **Activities**, represented by a rounded rectangle, denoting the tasks that need to be carried out.
  - **Gateways**, represented by a diamond shape, and can be used to express different decisions in a process as well as forking, merging and joining of paths.
- **Connecting objects**, represented by different types of arrows, which are used to connect flow objects together.

With these fundamental components, one can express the sequence of tasks involved in a workflow, including task branching and parallelism. Pools and swimlanes can also be used in a diagram to organise activities into separate visual categories, which can represent different actors in a system or different organisations interacting with each other- it is also common practice to not model external organisations (whose processes may not be known) by leaving their pool blank, and only using dashed connecting objects (arrows) to indicate messages being sent to and from the company.

The basic BPMN activities include **user** tasks (also known as human tasks), **service tasks**, **script tasks**, and **abstract tasks**. The choice of task type usually has a meaningful impact for BPM applications, however in BPS these are usually all treated the same.

The most common types of gateways include **parallel** and **exclusive gateways**. Exclusive gateways are used to model a decision or a fork in the workflow where only one of the available outgoing routes is taken, while parallel gateways model for parallelism, where every one of the outgoing paths is taken. Other types of gateways exist to, for example various types of event-based gateways, but these are slightly more rare [36].

The extended BPMN language has a large variety of expressive features, including a diverse range of intermediate events such as messages being received, timers, and errors, various types of tasks, sub-processes and call activities, event-based gateways, associations, data objects, and much more. This can lead to a lot of complexity and diversity, so the BPMN specification comes with some best practices, guidelines and examples. Despite this, as shown by M. zur Muehlen and J. Recker [36], BPMN diagrams are not complex in practice, as on average a BPMN model uses just 9 distinct constructs, with a large portion of models using only the most fundamental constructs such as tasks and sequence flows.

As shown by P. Wohed et al [35], BPMN lacks a lot of expressive power when it comes to process modelling for the purposes of simulation. Some workflow patterns cannot be represented, while others have multiple contrasting ways of being modeled, but the most notable shortcoming is the lack of support for resource representation. Pools and swimlanes, which represent parties and roles, are the closest BPMN comes

to expressing resource requirements, but the vast majority of resource patterns have no way of being shown.

2.3 Business Process Simulation Tools

There exists a wide selection of BPS tools, with varying capabilities and target users. A BPS tool is usually based on a modeling notation like Petri nets or BPMN. BPM tools are of particular relevance in this project, but some popular "general purpose" process simulation tools have also been considered below.

In the literature, the term "general purpose simulator" tends to refer specifically to *CPN Tools* [12] and *Arena* [13], and it seems to be used to distinguish them as tools which have been made for use in many disciplines outside of BPM, and as such they do not use BPMN. These are two of the most known simulation tools and they appear frequently in various surveys. CPN Tools is a general purpose coloured Petri net [11] framework with simulation functionality and Arena is a discrete event simulator which uses its own modelling language. Both are widely acknowledged as being strong simulation tools with good modeling and simulation functionality, but are sometimes critiqued as being hard to model with and "profound knowledge" of their modelling solution [10]. This is occasionally listed as a major disadvantage of these tools when compared to the more familiar BPMN-based BPS simulators [25].

There exist many business process simulation tools. Some of the most known tools include Adonis [7], BIMP [19], Bizagi [5], Bonita [6], and Visual Paradigm [16]. The tools listed here, like most BPM simulation tools, are BPMN-based. There are many differences in the capabilities of these tools [23, 25], but in general they tend to lack some of the features of the general purpose simulators, such as replications and built-in support for confidence intervals. The biggest advantage that these tools have over the general purpose tools like Arena and CPN is that they are easier to use and do not require "profound knowledge"[10] since they use BPMN. Unfortunately many of these tools are not open-source and do not have free versions, making them inaccessible for research and comparative evaluation. The common pattern is that these tools tend to originate as BPM tools which are extended with simulation capabilities, as opposed being purpose-made simulators [23], and this can result in poorer support for criteria like replications or resource roles .

Chapter 3

Proter

The Proter simulator was originally created as part of the **WorkflowFM** platform. WorkflowFM is a logic-based framework for formal process modelling and composition [21], and it consists of many subsystems ranging from the workflow modelling tool to a business process management dashboard. It uses a formal reasoning approach to process composition, resulting in correct-by-construction workflow models that can be executed asynchronously. WorkflowFM has been used in collaboration with manufacturing partners [22] and the NHS [2].

The simulation engine, Proter, is one piece of the WorkflowFM framework. Proter was once embedded as part of WorkflowFM, but in recent developments it has been extracted into a standalone unit, and made open-source. Proter is used to simulate user-defined models by concurrently running multiple discrete events in virtual time while respecting resource constraints. Given information about a workflow, such as the sequence of tasks involved, the duration and cost of tasks, and resource requirements, Proter can simulate the workflow by scheduling tasks to use the available resources for the duration that the task is live throughout virtual time. Proter then produces data files summarising the entire simulation, and a visual timeline which is helpful in visualising how the simulation transpired.

Proter is also a very unique simulator in that it places an emphasis on task priorities, whereby features such as prioritised scheduling of tasks and multiple resource allocation by priority are at the core of its functionality. In this chapter we give a brief overview of the Proter architecture and of these unique features.

3.1 Architecture

The basic software architecture of Proter consists of three key components: the **coordinator**, the **scheduler**, and the **simulation instance**. The interactions between these components are summarised in Figure 3.1.

In order to simulate a workflow, Proter needs information about the tasks involved. This includes things like the order of the tasks (control flow), the resources they need, their duration, and so on. This information is handled by the simulation instance,

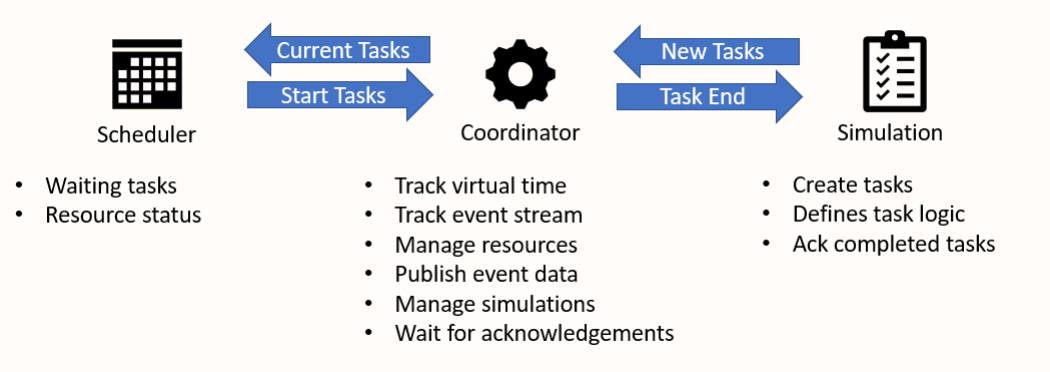


Figure 3.1: Annotation

which interacts with the coordinator by sending task information when it is needed. For example, when the coordinator messages the instance that a certain task has finished, the instance needs to then respond with the tasks that should start next. There can be many task instances during a single simulation, all talking to the same coordinator.

The coordinator primarily keeps track of virtual time through a stream of discrete events. In each timed event, it communicates with the simulation instances to find out which tasks are waiting to be started at the current virtual time and it attempts to start tasks by consulting the scheduler. Upon the event of a task finishing the coordinator can message the simulation instance in order to receive the next tasks. The coordinator has an internal event stream, and it progresses through virtual time by dealing with the events as they come up. The event stream can contain the following events:

- **`FinishingTask` events** - these events describe the time at which a certain task will complete. The task information is received from the simulation instance, and it begins with the permission of the scheduler, at which point the finishing task event is placed on the event stream to signify the time when the task will end.
- **`StartingSim` events** - this event contains a reference to a simulation instance. The coordinator will initialise the instance at the time of this event, therefore this enables us to plan for simulation instances to begin at some point in the future.
- **`TimeLimit` events** - this event marks the maximum duration of the simulation. If the time limit event is reached in the event stream, everything is forced to stop. Note that the simulation can also end if the event stream becomes empty, so a time limit event is not always necessary.

Finally, the scheduler determines which tasks can start given the current resource availability. It keeps track of which resources are busy or free, and by using this information it selects which of the currently available tasks should begin. If a task cannot start right away it is scheduled to start at some point in the future. The scheduler uses task priorities to intelligently schedule task starting times, and we discuss this in more detail in section 3.3.

3.2 Example Usage

The example shown below demonstrates a typical usage of Proter. Figure 3.2 shows a code extract in Scala which defines a simulation instance, and 3.3 shows the resource timeline which results from simulating this instance. This example uses the internal *Flows* structure to define the workflow, which is a tree-like structure loosely based on BPMN gateways, and it serves as a quick and fairly easy way to define simulation instances.

```
1  val task1 = new FlowTask(Task("task1",2L).withResources(Seq("r1")))
2  val task2 = new FlowTask(Task("task2",2L).withResources(Seq("r2")))
3  val task3 = new FlowTask(Task("task3",2L).withResources(Seq("r2")))
4
5  val flow = task1 > ( task2 + task3 )
6
7  val flow_sim = new FlowSimulation("flowSim",coordinator,flow)
8  coordinator.addSimulationNow(flow_sim)
```

Figure 3.2: Annotation

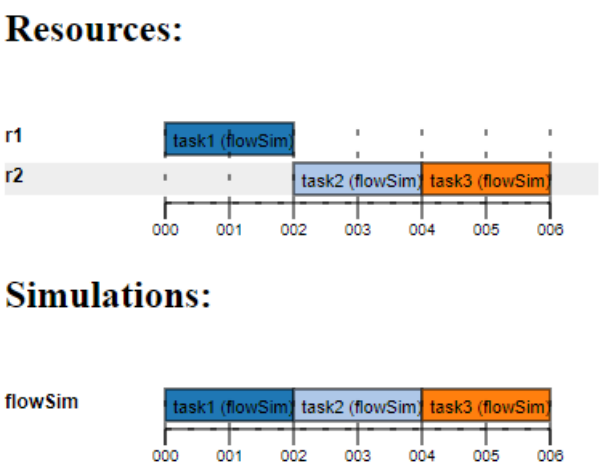
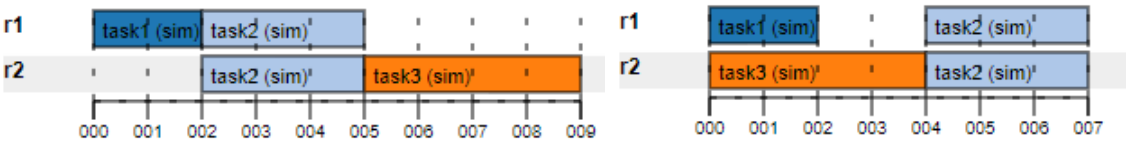


Figure 3.3: Annotation

On lines 1, 2, and 3 in the code extract we define tasks, which will become leaf nodes in the Flows tree. We can see that each task is given a name (e.g. "task1"), and a duration (in this case each task has a constant duration of 2). In addition, each task has a resource assigned to it, and resources are being referenced by their unique names (e.g. "r1").

Next, our Flow is defined on line 5. The tasks are combined into a single Flow using operators like "And" and "Then". In this example the sequence of tasks used is `task1 > ( task2 + task3 )`, which means that task 1 starts first, and once it completes both task 2 and task3 can begin in parallel. Internally, this would result in an Flows tree where a "Then" operator is the root node, with "task1" as the left child and an "And" operator as the right child. Finally on lines 7 and 8 we make a simulation instance from this Flow, and then send it to the coordinator.



(a) Timeline with prioritised scheduling (b) Timeline without prioritised scheduling

Figure 3.4: Resource timeline example with/without prioritised scheduling

Figure 3.3 shows the resulting timeline for this example. Because of the Flow we described it might seem that task 2 and task 3 should run in parallel after task 1 completes, but in reality they run in sequence. This is because both of these tasks use resource r2, so one task can start but the other must then wait for this resource to become available.

Only one simple instance is involved in this example, but in reality there could be many instances being simulated in the same environment, all of which compete for resources. Furthermore the tasks could have costs and durations drawn from probability distributions, different task priorities, and multiple resources.

3.3 Priorities and Scheduling

Priorities are at the core of Proter. As mentioned previously, the scheduler decides which available task can start in order of priority, and in case there is a conflict for a resource the task is scheduled to start as soon as the resource becomes available. If some high-priority task is scheduled at a point in the future to use a resource which is currently free, the scheduler knows that a low-priority task which uses that resource cannot begin unless it can finish in time, because this would result in further delaying the high-priority task.

The effect that this has is most clear with an example. Consider three tasks:

- Task1 has a duration of 2, requires resource r1, and has high priority.
- Task2 has a duration of 3, requires resources r1 and r2, and has medium priority.
- Task3 has a duration of 4, requires resource r2, and has low priority.

Given these, consider the flow `task1 + task2 + task3`, so all 3 tasks can run in parallel if not for resource conflicts. Figure 3.3 shows the result between the Proter prioritised scheduler (left) versus a more typical greedy first-come-first-served scheduler (right). With no consideration for priorities, the greedy scheduler finishes faster but task 2 is delayed by task 3 even though task 2 has a higher priority. With prioritised scheduling task 3 is delayed even though all of its required resources are free at the beginning, such that task 2 can start sooner. If task 3 was short enough that it could finish before task 2 could ever begin, the Proter scheduler would be able to slot it in.

Chapter 4

Evaluating Simulators

As mentioned in chapter 2, there exist many different simulation tools, and they can have vastly different capabilities and functionality. Naturally, selecting which tool to use will depend on the requirements that the user will have and the use cases of the simulator, yet one would think that the standard requirements of such simulation tools have been well defined. However, surprisingly, there is no clear-cut consensus on standard essential capabilities of BPM simulation tools in the literature. To our knowledge, only a handful of papers [23, 25, 10] attempt to survey some of the existing popular simulation tools, however the criteria which they use to evaluate tools varies a lot. Since some of the goals of this thesis include evaluating Proter alongside other simulation tools and implementing some important but previously missing functionality, it is imperative to first identify what simulator capabilities considered essential or expected, and to this end we rely on the existing surveys and some other papers on the topic of business process simulation.

4.1 Identifying Essential Capabilities

The first survey by M.H. Jansen-Vullers and M. Netjes [10] evaluates a number of tools using a wide variety of criteria. In contrast to our own focus, this particular survey is dedicated to evaluating the usability and presentation of the results in the tools, such as "animations" or the ability to replay simulations, with relatively little analysis of the technical capabilities of each tool. The most useful criteria as used by this paper include the importance of supporting standard workflow patterns, the resource and data perspective, and support for distributions.

The paper by Peters et al. [25] has a very useful analysis of simulation engines as part of their work on an unnamed prototype. They use a number of important functional criteria in their evaluation, including support of workflow patterns, distributions, resource perspective, and most notably a section named "Simulation" that includes warm-up periods, replications, and confidence intervals as advocated for by van der Aalst [30, 31] . Most of the criteria which they used is also applicable in this project, with the exception of the advanced resource constructs which they identify, such as queuing strategies, allocation strategies, and separation of duties. These advanced constructs are an

interesting perspective into the way resources are utilised in simulations but they appear to be selected specifically to bolster the prototype which they present, and the remainder of the paper goes on to focus on these constructs and the prototype. A very insightful takeaway from this paper is the comment about BPMN support: "General purpose simulation tools are less in favor than the business process simulation tools, which are easier to use and do not take a steep learning curve to model processes, since they use the BPMN modeling language". By "general purpose simulation tools" they refer specifically to Arena [3] and CPN tools [29, 12], which are not targeted at a specific industry unlike the business process management oriented tools which we consider most often.

The survey by J.L. Pereira and A.P. Freita [23] covers some similar topics to Peters et al., including the resource perspective and features such as replications, but they also evaluate other important criteria including context definition, arrival rates, and branch probabilities. Some of the criteria they cover, such as the inclusion of specific probability distributions, are useful from the perspective of an end user, however in the case of this thesis and from the stance of evaluating the tool's capabilities this level of detail is not needed, since we choose to focus on more general capabilities as opposed to specifics like the types of distributions supported.

Much of the work by van der Aalst presents a good foundation for business process simulation. In his Survival Guide [30] he goes into depth discussing the importance of sub-runs and a warm-up period for obtaining confidence intervals, and has an informative perspective on resources such as considering human actors who might have shifts, take days off, or have shifting levels of efficiency throughout the day. Van der Aalst also established standard workflow patterns such as parallelism and branching [33], which have been used to evaluate simulators in papers such as the one by Peters et al. [25].

From the literature discussed above, we have identified the following essential criteria:

1. **(Basic Flow) Sequence**, does the tool support the modelling and execution of a sequence of tasks?
2. **(Basic Flow) Parallelism**, does the tool support parallelism of tasks or sequences of tasks? The BPMN parallel gateway is the analogous modelling construct for this criterion.
3. **(Basic Flow) Branching** does the tool support branching within a workflow? The BPMN exclusive gateway is the analogous modelling construct for this criterion.
4. **Starting Time**, if the starting time of instances in the simulation is measured and recorded.
5. **Transfer Time**, if the transfer time if instances in the simulation is measured and recorded. This is the time spent "moving" the instance between resources, for example moving a part between machines in a workshop.
6. **Waiting Time**, if the waiting / queuing time of instances in the simulation is recorded. This is the time elapsed while a task is delayed due to some resource

being unavailable.

7. **Processing Time**, if the time spent processing instances is measured and recorded. This is the time that the tasks involved in the instance consume directly while they are completed, as opposed to the waiting time.
8. **Arrival Distributions**, if the tool supports the use of probability distributions to model the arrival rate of new instances into the simulation, for example using a negative exponential distribution to model the rate at which customers enter a store.
9. **Duration Distributions**, if the tool supports for task durations to be expressed using a probability distribution.
10. **Branch Probabilities**, if the tool allows the usage of probabilities to specify the likelihood of following different branching paths in the workflow.
11. **Resource Requirements**, if the tool allows for tasks to have resource requirements.
12. **Cost per Activity**, if the tool supports, measures, and records the costs associated with executing a task.
13. **(Resources) Capacity**, if resources in the simulation can have a certain capacity, for example if a machine in the workshop can operate on multiple items at once.
14. **(Resources) Roles** if the resources in the simulation can have roles, whereby a task may require a resource with a certain role as opposed to a specific resource. For example consider a workshop with three different lathes where any of these machines is equally suitable for a task requiring the use of a lathe.
15. **(Resources) Schedules** if the resources in the simulation can have schedules, for example the ability to model the shifts of workers.
16. **(Resources) Cost of Usage**, if resources can have an associated cost of usage, and if the simulation measures and records these costs.
17. **(Resources) Multiple Roles**, if resources can have multiple roles (see *(Resource) Roles*).
18. **(Simulation) Duration**, if the tool measures the duration of the simulation
19. **(Simulation) Warm-up Period**, if the tool supports the designation of a warm-up period in the simulation. A warm-up period is often included in simulation to account for the initial growth period before the simulation settles into a from of steady-state, where resources are still not fully used and the number of instances being simulated has not reached saturation.
20. **(Simulation) Replications**, if the tool allows the usage of replications, also known as sub-runs. This refers to either distinct re-runs of the simulation or the splitting of the simulation runtime into discrete chunks (the first of these could even be specified as the warm-up period). These are essential in obtaining statistically correct confidence intervals based on the central limit theorem.

Criteria	BIMP	Bizagi	BPSim	Proter
Sequence	+	+	+	+
Parallelism	+	+	+	+
Branching	+	+	+	+
Starting Time	+	+	+	+
Transfer Time	-	-	+	-
Waiting Time	+	+	+	+
Processing Time	+	+	+	+
Arrival Distributions	+	+	?	-
Duration Distributions	+	+	-	+
Branch Probabilities	+	+	+	+
Resource Requirements	+	+	+	+
Cost per Activity	+	+	+	+
Capacity	+	+	+	-
Roles	+	+	+	-
Schedules	+	+	+	-
Cost of Usage	+	+	+	+
Multiple Roles	-	-	+	-
Duration	+	+	+	+
Warm-up Period	-	-	-	-
Replications	-	+	?	-
Confidence Intervals	-	-	-	-

Table 4.1: Evaluation of simulation engines using essential criteria

21. **(Simulation) Confidence Intervals**, if the tool has built-in support for calculating the confidence intervals of Key Performance Indicators (KPIs) using the results of replications.

Table 4.1 evaluates some popular simulation engines, where + and – indicate if a feature or present or missing. The evaluation of these simulators is sampled from the surveys and other papers discussed previously, and the symbol ? is used in cases where these papers disagree on the presence or absence of a feature.

4.2 Initial Evaluation of Proter

Table 4.1 also includes an evaluation of Proter prior to any of the implementation involved in this thesis. A full and detailed evaluation of Proter after the implementation can be found in Chapter 7. As indicated in the table, Proter supports the standard workflow patterns (Sequence, Parallelism, Branching), as well as measuring various aspects of the simulation such as processing time and waiting time. Elements of randomness such as duration distributions and branch probabilities are supported, and the cost of a task and cost of usage of a resource can both be specified.

As expected, Proter had many missing criteria. The most important missing criteria gneraly fall under the category which Peters et al. [25] labeled simply as “Simulation”,

which encapsulates capabilities such as Warm-up Period, Replications, and Confidence Intervals. In reality the problem in Proter is a bit more deep-rooted than simply missing these features: Proter does not have an arrival process at all.

Van der Aalst et al. [31, 30] present the arrival process as an absolute essential, and it is the foundation of their argument for the importance of the other criteria such as Warm-up periods and replications. The arrival process describes the rate at which new simulation instances enter the system, such as the rate of customers in a store or the rate of new jobs in a manufacturing facility. In past work with WorkflowFM many instances of a simulation would be set to start at different times, and this could be computed externally to the Proter simulation engine. Within Proter, however, simulation instances can only be assigned individually at specific times, and there is no process for assigning instances at a set rate or even for generating them. Since there are no arrival processes, features like warm-up periods and replications cannot be implemented. Also for this reason the "Arrival Distributions" criteria is not met. This feature stands out as a very common and essential requirement of simulation tools, and in this regard Proter falls behind when compared to other business process simulators.

The other major missing capabilities include resource capacity, roles, and (resource) schedules. These, and other resource-related criteria, are sometimes referred to as the resource perspective, and this is a very interesting area of simulation which is often overlooked or underestimated by simulators[30]. much of the "resource perspective" criteria are very desirable in a simulator and we will return to tackle this topic in future work, however the above criteria were selected as the focus of this project due to how essential they are for simulation. In particular the arrival process is absolutely indispensable, given how implicit it seems to be in the literature and the fact that all other simulation tools have this functionality.

One other feature which is very widespread and seemingly indispensable for business process simulation is support for BPMN models. The Survival Guide [30] states that common modeling notations such as BPMN are preferred over proprietary solutions due to their familiarity, Jansen-Vullers and Netjes [10] state in their survey that BPMN simulators are proffered since they do not require "profound knowledge" to be used, and Peters et al. [25] point out that simulators using BPMN are preferred because they are easier to use and do not require a steep learning curve. Due to this massive popularity of BPMN models, and due to the fact that they are supported by the majority of business process simulators, we have decided that this is another essential component which must be added to Proter so that it is more accessible to a BPM audience, and easier to compare against other existing tools.

In the next two chapters we discuss the implementation of the essential criteria identified above, and the support for BPMN models.

Chapter 5

Supporting Essential Capabilities

As we observed in Chapter 4, there are a number of criteria which appear to be considered essential throughout simulation tools, yet are missing in Proter. As part of the goal of this thesis we set out to implement the most essential of these missing criteria with the aim raising Proter to be on-par with the general expectations for a business process simulator as seen throughout the literature.

Based on the missing criteria as shown in Table 4.1, the following criteria were originally selected for this project as being the most essential:

1. Arrival Process
2. Arrival Distributions
3. Replications
4. Warm-up Period
5. Confidence Intervals

Notice that the "arrival process" feature is not an item in Table 4.1, since curiously the existing literature does not include this in their own evaluations - perhaps they thought it was implicit for all business process simulators - however it is clear that this is an absolute essential and this feature forms the foundation for all the other elements in the above list.

The above criteria are presented in order of importance, and they were worked on in this order during the project. We prioritise the implementation of the essentials first, followed by the most common criteria. This order is based on the frequency with which other simulators implement the criteria. From the literature discussed previously we find that confidence intervals and warm-up periods are particularly rare among business process simulators. A potential justification of this is that these can be calculated externally by the data analyst given the output of the simulation. Additionally there is a large breadth of potential KPIs which the analyst may be interested in, many of which might require additional calculations prior to obtaining confidence intervals and so supporting KPIs in anticipation can be extremely challenging and support for an exhaustive set of KPIs might be impossible.

5.1 Arrival Process and Distributions

5.1.1 Design

Within Proter singular simulation instances are typically added to the coordinator using one of its methods such as `addSimulation` or `addSimulationNow`. These methods have the effect of placing a new event on the coordinator's event queue, which is responsible for starting a new simulation case. Once the coordinator eventually reaches this event, it will communicate with the simulation instance, sending and receiving information about the tasks involved as virtual time progresses, thereby simulating this instance. For an arrival process, we would like to maintain the same pattern of interaction with the coordinator, such that we only send information about the process to the coordinator once at the beginning and then the coordinator interacts with the process itself during simulation.

The arrival process needs to consist of the following:

1. An arrival rate.
2. A simulation instance generator.

The arrival rate, also known as the arrival distribution, will need to return the length of time between two separate instances from the arrival process, and the simulation instance generator will be responsible for providing these fresh instances for the coordinator to use.

5.1.2 Implementation

At its core the arrival process is a new event which can appear in the coordinator's event stream. This event extends the existing `DiscreteEvent` trait and consist of the time at which it occurs, the arrival rate associated with the process, and the simulation generator. Similar to other events in Proter, it is eventually reached and handled by the coordinator once virtual time progresses. When this happens, other events tend to call a certain method before being removed but the arrival process event is self-replicating, in that by handling this event a new identical arrival process event is placed onto the coordinator's event stream.

```
1  case ArrivalProcess(t, rate, simulationGenerator) => {
2      events += ArrivalProcess(rate.next(t).round, rate, simulationGenerator)
3      addSimulationNow(simulationGenerator.newSim(this))
4  }
```

Figure 5.1: Code extract showing how the arrival process event is handled in Proter

An extract of code showing how the arrival process event is handled by the coordinator is shown in Figure 5.1. Our new event consists of a time t, an arrival rate, and a generator, as seen on line 1. We make a duplicate of this event which is placed back onto the event stream on line 2, and this new event has a new time which is sampled from the arrival rate, so through this process subsequent events will occur at different future times. Lastly, on line 3 we start a new simulation instance at the current virtual

time. The simulation generator associated with this arrival process provides the new simulation instance, and from here on out the coordinator and instance proceed to interact as usual.

The `rate` in Figure 5.1 refers to an instance of the new `ArrivalRate` class, and is used to obtain the time at which the next event should occur using the `next` method. Random numbers can be used in order to sample from a random distribution, for example in the following equations which show how the result in equation 5.5 was obtained which allows us to get a time interval between the arrival of two instances as modeled by the negative exponential distribution function:

Density function:

$$f_x(t) = \lambda e^{-\lambda t} \tag{5.1}$$

Cumulative distribution:

$$F_x(t) = 1 - e(-\lambda t) \tag{5.2}$$

Given a random number r, $0 \leq \mathrm{r} \leq 1$:

$$r = F_x(t) \tag{5.3}$$
$$\Rightarrow t = -ln(1-r)/\lambda \tag{5.4}$$
$$\Rightarrow t = -ln(r)/\lambda \tag{5.5}$$

The negative exponential distribution is the most common distribution used for simulation due to its link to its ties to possion arrival processes and fitness to human patterns, however other distributions, such as normal or gamma, are sometimes used too.

The new `SimulationGenerator` class provides an interface which is implemented by various sub-classes such that given necessary data in the constructor they are able to return new simulation instances. A handful of basic simulation generators are implemented for basic single-task, flow, and BPMN (see Chapter 6) simulations are provided, but one may require more complex behaviour from their generator, such as generating different instanced depending on the current time or congestion, in which case they would need to implement the interface themselves. The one important thing to remember is that the names of every instance generated by the generator must be different since they are used as unique identifiers within Proter. The provided generators do so by keeping an internal counter and appending its value to the base names which are specified by the user.

5.2 Replications

In Proter, simulation metrics are handled using a publisher-subscriber pattern between the coordinator and a simulation metrics output handler. The coordinator publishes a

message for all simulation metrics, which includes things such as a simulation instance starting, a task finishing, or when a resource is being used by a specific task. The simulation metrics handler then receives all of these events and records each one as needed. The top-level handler distributes event information to various output handlers and aggregators, which can have functionality such as saving information to a csv or printing a summary at the end of the simulation.

To implement replications, we have chosen the approach of splitting a longer simulation into multiple sub-runs. The alternative, according to the Survival Guide [30], is to run a shorter simulation multiple times, but we have chosen the prior technique because it is easier to implement into Proter, since we can simply adapt the simulation metrics handler functionality to store event information into separate bins depending on the virtual time.

This implementation introduces a new metrics handler which contains multiple metrics aggregators. Each aggregator is responsible for a separate sub-run of the overall simulation. When an event is received by this metrics handler, it inspects the virtual timestamp to decide which aggregator should receive the event. The aggregator number is determined using this formula, where n is the total number of replications and therefor the total number of aggregators:

$$\lfloor \frac{timestamp}{totalTime \div n} \rfloor$$

Special care has to be taken with certain events to avoid incomplete information in old aggregators. For example it would be possible that an aggregator is told a simulation instance has started, but by the time the it finishes the next aggregator is being used, and so the initial aggregator would never find out that the instance has completed. This is why events such as simulation instances ending or resources being added are forwarded to all aggregators.

5.3 Warm-up Period and Confidence Intervals

Unfortunately warm-up periods and confidence intervals were features which which we were not able to finish. This was due to time constraints which resulted from troubles during the implementation of BPMN into Proter, which was a more important milestone in this project. Luckily because of the prioritised approach to the criteria which we set out to implement, we managed to successfully implement the other, more essential features. We can find additional solace in the fact that warm-up periods and confidence intervals are also not supported in any major business process simulators. It seems to be a common practice in BPS studies to carry out this sort of data manipulation outside of the simulation tool, especially since the KPIs being studied can sometimes require additional calculations to obtain, for example if the study is interested in median queuing times while the simulator only calculates the mean.

Chapter 6

BPMN Integration

There is a clear affinity for BPMN within the world of business process management. We saw a hint of this in the paper by Peters et al. [25] in Chapter 4, where they say that tools which do not use BPMN are "less in favour" due to a steeper learning curve. From the tool surveys which were examined previously it is also apparent that the vast majority of simulation tools use BPMN. It is clear that, despite its shortcomings for simulation purposes, BPMN is very popular and widely used among business process management professionals, and so as part of our goal to establish Proter as a state-of-the-art business process simulation tool it is essential to support BPMN in order to be an approachable and attractive option for researchers and professionals alike.

6.1 Design

Proter requires that an instance of a class implementing the `Simulation` trait is provided to the coordinator so that it can be simulated. There is a small number of built-in ways to create simulation instances, such as the recently developed *Flows* notation which provides a simple way of expressing the sequence of tasks. In order to support BPMN we need to take a `.bpmn` file as input and construct a subclass of `Simulation` which can communicate the contents of the file to the coordinator.

Per the original design it was intended that the BPMN model would be processed using an external library such as the popular jBPM [8], however unfortunately this did not work for reasons discussed in Section 6.2. Instead in the final design we parse the BPMN file ourselves in order to figure out the sequence of tasks in the workflow.

Since BPMN only expresses the sequence of tasks in the workflow (control flow), additional information will need to be provided to create a full model as is also standard among other simulation tools. In this design we will pass an additional configuration file which stores necessary information on task cost, resources, duration, and priority.

6.2 Challenges

As mentioned in Chapter 2, BPMN is a very intricate and verbose notation. It was not created for simulation or programmatic usage, but rather for communicating business workflows between management professionals. Because of this, a model can consist of a wide variety of events, gateways, and activities, not to mention black box pools and sub-processes which make simulating a model even harder. Supporting all the possible constructs and all the ways in which they could connect is a significant challenge.

Due to this complexity, it is desirable to use a mature external library, such as the relatively popular jBPM [8], that can already parse and execute BPMN models. Initially we planned to use the BPMN engine in jBPM to step through the model in parallel to the Proter simulation. This would work by starting the jBPM engine at the beginning of the simulation instance's lifespan, and whenever a task finished according to the coordinator we would report that the corresponding BPMN activity has completed to the jBPM engine, hence obtaining the next tasks from the engine and sending those to the coordinator. In this design, the simulation instance would be an intermediary between the coordinator and the jBPM engine, where it would receive activities from the engine, create Proter tasks by pairing activities with the corresponding cost, duration, and resource information, and sending them off to the coordinator.

Unfortunately, this did not work out even after a considerable amount of effort. Disappointingly, BPM libraries such as jBPM are designed in a way which locks you into using their entire ecosystem, without much room for utilising specific tools or components. In jBPM, the BPMN engine is tightly coupled with multiple other jBPM components and cannot work on its own, and trying to execute each type of task came with its own complications:

- Abstract tasks completely do not work since one cannot control when they are completed and the engine essentially skips them.
- Script tasks seem promising since they allow us to call any function in the code, but they execute automatically (without waiting for us to start them) and they are blocking even if they are marked as "asynchronous" which means that even if we run them on a separate thread they will prevent us from retrieving other pending tasks from the engine until they finish processing.
- Human tasks need to be provided with a registered user that belongs to the correct department in order to retrieve them from the engine and to complete them. Even once they are retrieved there are difficulties with finding which tasks are pending.
- The documentation for this is very poor, and there exist a few examples which use the engine but they are designed to be implemented with the rest of the jBPM system and there's not much to work with in our direction.

In general jBPM was very hard to work with, and because of poor documentation our efforts were riddled with much trial and error. We tried to use other BPM tools, such as Camunda [15], but this also did not work out. Unfortunately it turns out that despite how easy it is to execute BPMN within the ecosystem of these tools, it is very difficult

to integrate the BPMN execution component with custom external software.

Because of this, we had to resort to stepping through the BPMN model ourselves. Naturally this poses its own challenges, and it ties back to the fact that BPMN is fairly verbose, with a large number of variations of activities, events, and gateways. Supporting all of these variations offered by BPMN would be desirable, but is a potentially long and arduous process which would arguably not add much value to Proter. In addition, it is known that despite the available variety, on average only 9 distinct constructs are used in a BPMN model [36].

1. Basic BPMN Activities: Abstract, User, Script, and Service Tasks
2. Exclusive Gateways
3. Parallel Gateways

These, plus start and end events, are among the most commonly used BPMN constructs [36], the remaining ones being pools and swimlanes, however these are typically used to model actors or companies, which we would instead model with resources in a simulation.. Other more complex constructs, such as events or message flows would be a nice addition to the supported constructs, but due to the limited time for this project we did not consider them as essential.

6.3 Implementation

The implementation uses the Camunda [15] library to help with parsing the BPMN model. As mentioned before, we tried using the BPMN engine in Camunda to execute BPMN models for us, but this did not work. Instead we ended up using the parser included in this library, which allows us to easily query the XML-based model, but the execution of the model is down to our own implementation.

A new `BPMNSimulation` class was introduced as a subclass of `Simulation`. Other than a name and a reference to the coordinator, an instance of this class requires the path to the model `.bpmn` file and to the model data `.json` file. These are used to create an instance of the new `BPMN` class.

The json file is translated into a `BpmnDataset` data structure, which stores a list of entries describing the cost, duration, resources, and priority of tasks in the BPMN model. This is stored in the `BPMN` class alongisde a Camunda `BpmnModelInstance`. The Camunda model instance allows us to traverse the BPMN model in order to find the sequence tasks or gateways that appear, for example given the id of a task we can find the tasks or gateways that are connected after it in the diagram.

The core usage of the new `BPMNSimulation` class revolves around reacting to a previous task completing in the coordinator. The following sequence occurs each time a task completes according to the coordinator:

1. The id of the BPMN object corresponding to this task is found. A map from Proter task ID to BPMN object ID is maintained making this easy.
2. The id of the task that just completed is removed from the above mentioned map.

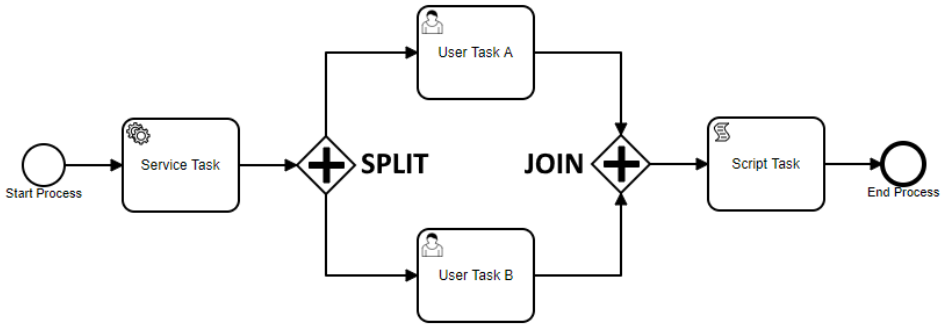


Figure 6.1: An example BPMN model with a split and join parallel gateway

3. We find all of the next BPMN objects (tasks or gateways) that appear in the model after the task with the corresponding id.

4. For each next BPMN object:

   (a) If this is a task (user task, script task, abstract task) make a Proter `Task` using the information found in the corresponding model data entry. This task is sent to the coordinator and a mapping from Proter task id to BPMN object id is saved for later.

   (b) If the object is an exclusive gateway, one of the possible outgoing paths can be taken. A decision is made randomly between the possible options and then step 4 is repeated for the object(s) which come next.

   (c) If the object is a parallel gateway, all outgoing routes are taken but we need to make a distinction between "split" and "join" parallel gateways: A split has one input and many outputs, while a join has many inputs and a single output. An example of this is shown in Figure 6.1.

   For split gateways we simply execute every outgoing path by repeating step 4 for every branch. For join gateways we must wait until all of the input paths have completed, since otherwise we would execute the output path every time one of the input times reaches the gateway. In the example from Figure 6.1, the join gateway should wait until the top and the bottom branches both reach it before the script task can be executed. We achieve this by keeping track of "tokens" at each input of the gateway in a fashion inspired by Petri nets [4]. Every time a branch arrives at the input of a join gateway a token is placed at this input. When all of the inputs have at least one token, the gateway will consume one token per input branch and execute the output branch by repeating step 4.

The Petri net style approach to parallel join gateways described in step 4c ensures that loops and complex workflow patterns will work with this gateway. This functionality is achieved with a `ParallelGateway` class which stores the number of tokens at each input, and a map between BPMN gateway id and corresponding parallel gateway objects is maintained in `BPMNSimulation`.

```
{
  "data": [
  {"id": "serviceTask", "duration":{"type":"ConstantGenerator", "value":5},
      "cost":0, "priority":"Medium", "resources":["r1"]},

  {"id": "userTaskA", "duration":{"type":"ConstantGenerator", "value":5},
      "cost":0, "priority":"Medium", "resources":["r2"]},

  {"id": "userTaskB", "duration":{"type":"ConstantGenerator", "value":5},
      "cost":0, "priority":"Medium", "resources":["r2"]},

  {"id": "scriptTask", "duration":{"type":"ConstantGenerator", "value":5},
      "cost":0, "priority":"Medium", "resources":["r3"]}
  ]
}
```

Figure 6.2: An example json model data file

The model data structure mentioned throughout this section is built from the input json file with the help of the Scala spray-json library [28]. Custom json formatting objects were made by extending spray-json classes, which allow us to encode and decode Proter data structures. This is what lets us build instances of the `BpmnData` data structure from the json file.

An example of a json model data file used in this project is shown in Figure 6.2. Such a file must consist of a list of objects named "data". Each object in the list refers to a single task in the BPMN model corresponding to this file. The entry has the id of a task, the duration (expressed using a Proter value generator), the cost, priority, and the list of resources used by the task.

Using the json and BPMN files we can create`BPMNSimulation` simulation instances and send these to the coordinator, and from there the coordinator and instance can interact with each other as normal. For example, we can simulate the model in Figure 6.1 with the json in Figure 6.2, which generates the timeline shown in Figure 6.3. Note that user Task A and User Task B happen one after another despite the parallel gateway because they are competing for the same resource.

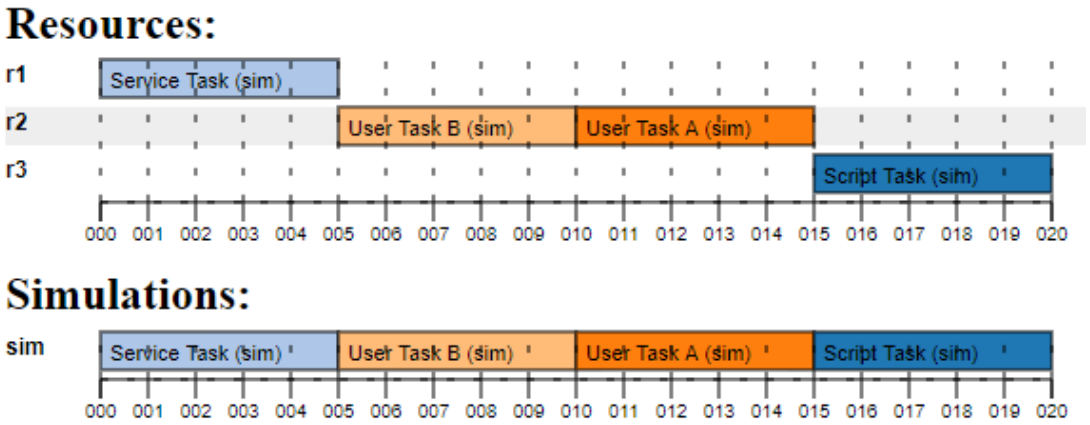


Figure 6.3: An example output timeline generated by using a BPMN model

Chapter 7

Evaluation of Proter

Having now implemented some important functionality and BPMN compatibility into Proter we will now look at a comparison between Proter and some other popular BPS tools, our goal being to demonstrate that the implementation is correct and that we get the same results given the same inputs. Thanks to the new BPMN support we are able to use the exact same BPMN model in each tool, so that the only differences in results will arise from the simulators themselves. Doing such a comparison also lets us discuss the ways in which these tools differ both in terms of functionality and in terms of the way in which they support the "essential criteria" which we identified in Chapter 4.

After this comparison we will also examine the benefit of some of the unique features offered by Proter, namely prioritised scheduling and look-ahead. These are not supported by any of the tools which we examine, and to our knowledge no business process simulators implement similar behaviour despite the popularity of similar topics such as job shop scheduling in the surrounding literature.

7.1 Comparison with Existing Tools

In this evaluation we compare Proter to the **Bonita simulator** (community version 6.5.3) [6] and **BIMP** (online academic version) [19]. These are both popular BPS tools which have featured in a number of the surveys which were analysed in Chapter 4. These simulators are also easily accessible unlike many alternatives such as Bizagi [5] and Visual Paradigm [16] which cost money making them inaccessible for research given the scope of this project. We also tried other freely available tools such as Bizagi [5] and Scylla [1], but we were unable to obtain functional simulations in them. It should be noted that we are using an old version of Bonita since unfortunately the latest community version of this tool no longer supports simulation.

There is no standard dataset or procedure for doing such an evaluation documented in any of the literature which we reviewed. As such, a total of 23 unique examples were made from scratch for this evaluation. These consist of a BPMN model and auxiliary information about the configuration used for the simulation which includes the arrival

rate, total execution time / number of instances to simulate, resource information, and task information. These were made to test all of the simulation criteria which we have identified in Chapter 4 as well as some interesting simulation properties as possible. This includes examples of sequences of tasks, parallelism, branching, loops, interesting resource patterns, such as deliberate resource conflicts between tasks, and interesting time properties such as a "tight fit" between the arrival of two instances versus long intervals.

BPMN models used in these examples are mostly purpose-made, while a few have been taken from example repositories of Camunda and Scylla. The problem with most BPMN examples available online is that they were not made for simulation purposes and they tend to contain many complex constructs (events, advanced gateways, black-box swim lanes) which impede the process of adapting them for a simulation.

A reference for every single example is included in the Appendix, and the examples themselves are submitted with this thesis. These references are simply short descriptions of the BPMN model and its environment, for example "1. (Basic Flow) Sequence, all one resource" or "13. Simple loop - one task".

7.1.1 Obtaining Results

Each example from our dataset was run on all three simulators. A script has been created to execute all the examples automatically in Proter. For the other simulators, however, the configuration has to be set manually each time. This process is made slightly harder by the fact that the different tools have different requirements for how some information has to be formatted:

- **Proter** - The simulation parameters consist of the arrival distribution used and the time limit for the simulation. This means that if our example calls for 100 instances to be simulated using a uniform arrival distribution with 20 time units between instances, then we need to set a time limit of 2000 units in our simulation.
- **Bonita** - Bonita uses Load Profiles to define an injection period, which has to be specified by selecting the date and time from a calendar (see Figure 7.3). Using the same example of 100 instances to be simulated with 20 hours between each instance, we would set the injection period to start on January 1st at midnight (00:00) and end on March the 24th at 8am with a uniform distribution and 100 instances, which works out to exactly 2000 hours. The process of figuring out the date and time which is needed and then selecting it from a calendar interface is quite cumbersome.
- **BIMP** - Similarly to Bonita, in BIMP we need to supply the number of instances to simulate. However this time we specify the inter arrival time between instances. Using our example from before we directly type in 100 instances and 20 inter arrival time.

The tools also vary significantly in the output which they generate, so it is not trivial to select the values by which to compare the three. With Proter all the information

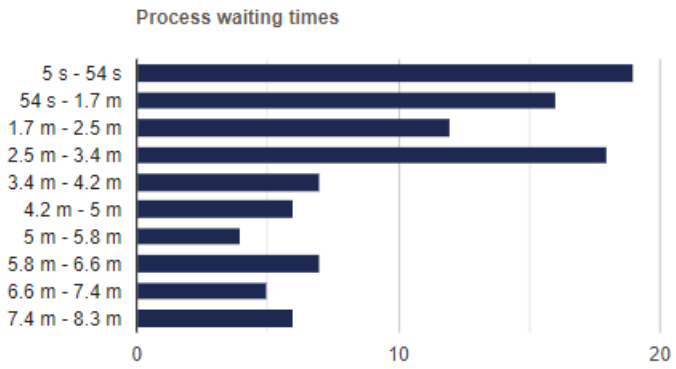


Figure 7.1: Example of a BIMP bar chart output

about resources, tasks, and simulation instances is stored in csv files and we are able to process this data to extract any desired value. Bonita has an option to export simulation details, but the csv files that it generates are empty so this is presumably a broken feature. Thus, the only information we can access is given in the report generated at the end of a simulation, which contains minimum, maximum, and average values for properties such as task duration and resource utilisation. BIMP similarly generates a report with some minimum, maximum, and average values for measurements such as task costs or simulation instance times. However, a lot of the data is shown in bar charts using bins as in the example in Figure 7.1, which makes getting some specific values a tedious manual process.

Ultimately these values have been recorded from each tool and each example scenario, because they give a good indication of how the simulation went and are obtainable from all three tools:

- Resource utilisation
- Average instance waiting time
- Average instance execution time

As mentioned, the BIMP output for instance waiting times is divided into bins as shown in Figure 7.1, so the values which we used in the final table have been calculated manually by using the count and average value of each bin, but unfortunately this is just an estimate of the true average instance waiting time.

The final results table is fairly large and can be found in the Appendix. A summary of the results is shown here in Table 7.1. The summary table shows the difference in values between Proter and the other tools, and the utilisation of individual resources has been merged into an average resource utilisation column. Note that multiple Bonita trials did not finish or did not work for reasons we will discuss in the next section, and these have been highlighted gray in the summary table.

	Proter-Bonita Difference			Proter-BIMP Difference		
Example	Avg. Resource Utilisation (%)	Avg. Waiting Time	Avg. Execution Time	Avg. Resource Utilisation (%)	Avg. Waiting Time	Avg. Execution Time
1	0.175	0	0.1	0.0475	0.5	0
2	0.175	5	0.15	0	0.5	0
3	0.4	0	0	0.06	0.5	0
4	0.3	5	0	0	0.5	0
5	0.175	0	0.1	0.0475	0.5	0
6	7.05	0	0	0.53	0.5	0
7	0.4275	0	0	0	0.5	0
8	0.2775	5	0.05	0.0825	0.5	0
9	0.2775	0	0.05	0.1275	0.5	0
10	0.2325	4	0.07	0.1	0.5	0
11	0.1525	5	0	0.14	0.5	0
12	0.2025	0	0.05	0	0.5	0
13	7.4325	1.7	6.2	0.22	0.12	0.1
14	N/A	N/A	N/A	9.09	1.89	5.15
15	0.075	0	0	0.0675	0.5	0
16	5.785	0	0.1	0.8525	0.5	0
17	0.275	0	0	0.0475	0.5	0
18	24.838	5.4	6.3	0.665	2.5	1.6
19	N/A	N/A	N/A	0.25	1.02	1.5
20	N/A	N/A	N/A	0.6925	0.5	0.45
21	0.3825	2.11	0.99	1.065	0.13	0.19
22	9.4825	8.63	7.92	2.9525	14.93	5.83
23	6.565	69.37	33.38	2.485	116.44	58.57

Table 7.1: A summary of the evaluation results

7.1.2 Discussion of the Results

Probably the most apparent thing from the summary table is that most of the results are very similar, and so the difference in the measured values is almost always nearly zero. This is good news as it shows that Proter gets the same results as these simulators and it is good evidence that the features implemented in this project work as intended. The reason why the values are not exactly equal (and therefore why the difference between them shown in the summary table is not always exactly zero) can be explained, and this reveals some interesting differences between the way in which the simulators work.

First let us look at the **average execution time**. Initially it might seem that the measured time is meant to be identical, however it should be expected that the value will be slightly different if the model has some element of randomness involved, such as loops which can be taken with a certain probability, or probability distributions for task du-

rations or the arrival rate. Only about a third of the models prepared for this evaluation have such random elements, namely examples 13, 14, and 19 to 23. This explains the discrepancies between Proter and BIMP, where we can see that the examples with no randomness yield identical average execution times (hence the difference is zero in the summary table), and for the other examples the result varies slightly.

For Bonita, the examples with no randomness are still slightly different. This arises due to a quirk in the way that Bonita works: it seems that no matter what we tried there was always an outlier instance which was somehow delayed and had a longer duration than the rest. For example, in the very first model we have a sequence of three tasks, each of which lasts 5 hours. There is no randomness, so the duration of every single instance should be exactly 15 hours. Proter and BIMP both report this result, but in Bonita report we find that exactly one instance always lasted 25 hours. This bumps up the average instance duration to 15.1 hours, hence the 0.1 difference between Proter and Bonita in the summary table. The reason for this behaviour is unclear but it could be a flaw to do with the way Bonita implements its load profiles. This explains why the Proter-Bonita difference in average execution time tends to be slightly off even in cases where Proter and BIMP agree on the result.

Now we need to mention the problematic examples in Bonita, which have been highlighted gray in the summary table and red in the full results table in the appendix. In the examples with **N/A** (or **DNF** in the full results) Bonita would either start trying to simulate the model but never finish, or it would throw a null pointer exception or "cycle detection" error. The other examples, which are highlighted but still contain readings, went wrong for some other reason or could not be fully modelled.

- **Example 13** contains a loop, but it is clear from the results that the loop was never taken.
- **Examples 14 and 20** contain loops and the Bonita simulation fails.
- **Example 19** is a complete anomaly. It seems perfectly normal (no loops or complex patterns, just two tasks in sequence with an optional third task in-between connected by an exclusive gateway) but a null pointer exception is thrown. It should be noted that the exact same BPMN model works just fine in BIMP and Proter, and it's completely unclear what causes the error.
- **Examples 21, 22, and 23** require the use of duration distributions and arrival rate distributions. Bonita does not have arrival rate distributions other than a uniform distribution, and despite having an option for specifying some sort of range for the duration of a task as shown in Figure 7.2, we found that no matter what input is given the task always runs for the minimum duration of time (e.g. in the example figure this task always lasts for 5 hours despite the 50% and 100% estimate and max time parameters). These examples were still carried out but the Bonita parameters are only approximations for what is called for by the model descriptions of these examples, so the results are inaccurate.

The issues we encountered in examples 13, 14, 19, and 20 might possibly be due human error. This is due to the limited user interface, options, and documentation of Bonita which make it very hard to work with, especially seeing as many of the solu-

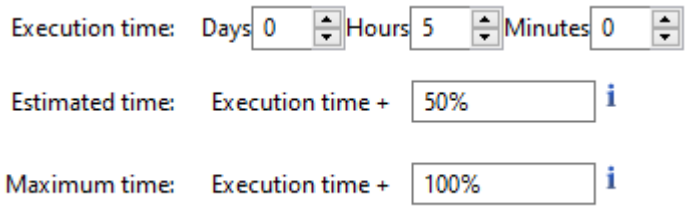


Figure 7.2: Bonita Simulator task execution time interface

tions they use are non-standard such as the injection periods and sampling intervals. Nonetheless, BIMP confirms the right results.

Getting back to the summary results table, let us consider the discrepancies between Proter and the other simulators in the **average resource utilisation**. For both Bonita and BIMP, the difference with Proter is almost always extremely close to zero, but almost never exactly zero. The difference in reported utilisation tends to be about 0.2% off, and is predictably larger in cases with more elements of randomness, such as examples 22 and 23. This small difference is easily explained by the way the Proter arrival process works.

In Proter we specify a time limit, and the arrival process continues adding new instances into the system until the time limit is reached, but in Bonita and BIMP we specify the number of instances to simulate. For example, if a model description in our dataset says to simulate 100 instances with a uniform arrival rate and 20 hours between arrivals, we set a time limit of 2000 in Proter. In this example, consider what might happen if one instance completes in exactly 15 hours (as is the case in example number 1 in the dataset). In Proter the 100th instance would start at time 1980 hours and finish at 1995 hours, then there are 5 hours of no instances and then the simulation terminates at 2000 hours. In contrast, in a simulator like BIMP the 100th instance also finishes at 1995 hours but then since all 100 instances have been simulated as requested in the input, the simulation terminates at 1995 hours. In both cases the resources were used for the exact same amount of time (e.g. example 1, resource r1 is used for exactly 1500 hours in both Proter and BIMP), but due to the different total simulation durations the resource utilisation percentage is slightly different.

This is exactly the reason for the discrepancies between Proter and BIMP, but then why is utilisation reported by Bonita slightly different from both the one reported by BIMP and Proter? This is possibly linked with Bonita's injection period and sampling interval parameters. In Bonita we specify the number of instances to simulate and an injection period. Since we cannot control the specific inter-instance times the specifics of when they actually begin is out of our control. It might seem as though you could calculate the time in-between instances given the injection period, however this is further complicated by the fact that the true duration of a simulation differs from the one specified in the injection period. Returning to our running example, we want 100 instances simulated over 2000 hours such that the instances are uniformly distributed with 20 hours between each arrival. Knowing this we define an injection period as shown in Figure 7.3, using the fact that there are exactly 2000 hours between Midnight

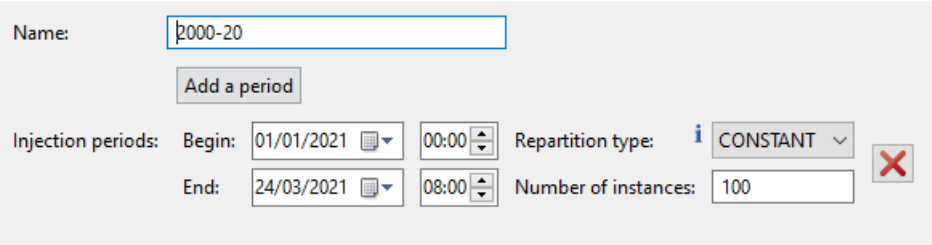


Figure 7.3: Bonita Simulator Load Profile interface, used for defining the injection period

on January 1st and 8am on March the 24th, which is 83.33 days. Despite this, and despite the fact that we know for certain that an instance in this simulation lasts exactly 15 hours, the simulation duration reported by Bonita in the final report is 82 days 13 hours 14 minutes 24 seconds. This result is very enigmatic, but one idea is that this is linked with the way Bonita uses sampling intervals, but even still if the sampling interval is set to 6 hours and a simulation instance lasts 15, and the total injection period duration is 2000, then why does the simulation duration involve minutes and seconds? This remains as another Bonita mystery, but what's clear is that the simulation duration is different from that which we specify, and so resource utilisation percentage reported will also, of course, differ slightly from the value which we expect.

Lastly let us discuss the **average waiting time**. As mentioned previously, BIMP does not report this value directly, but instead it produces a bar chart which summarises the range of waiting times as shown in Figure 7.1. We therefore need to estimate the average waiting time by taking the median value of each bin and scaling by the number of instances in that bin. This means that the reported value will be slightly off from the true average even for simulations with 0 waiting time, since in these cases BIMP presents a single bar in the bar chart with the range 0 to 1, thus we record 0.5 in the results. For Bonita, we were unable to get the waiting time to work at all.

Surprisingly, for every single example we tried Bonita reports 0 waiting time even if it is abundantly clear that there is lots of queuing and delays in the model. Many different inputs were tried, but the result never changed, so we suspect this is a broken feature. As a result, the values you see in the summary table are simply Proter's measured waiting times for each example.

Overall it is clear that Proter works as well as any of these other tools. We have shown over a variety of examples that the behaviour of Proter is correct, and that the small differences in the results arise from the very different ways in which each tool works. These are important differences which are hard to discern at-a-glance or from reading a tool survey, but as shown they have a noticeable effect on the output of the simulators. Despite these differences it is clear that the new functionality which we have added to Proter, which includes the "critical" simulation capabilities and BPMN support, both works and produces correct results.

Criteria	BIMP	Bonita	Bizagi	BPSim	Proter
Sequence	+	+	+	+	+
Parallelism	+	+	+	+	+
Branching	+	+	+	+	+
Starting Time	+	+	+	+	+
Transfer Time	-	-	-	+	-
Waiting Time	+	-	+	+	+
Processing Time	+	+	+	+	+
Arrival Distributions	+	-	+	?	+
Duration Distributions	+	-	+	-	+
Branch Probabilities	+	+	+	+	+
Resource Requirements	+	+	+	+	+
Cost per Activity	+	+	+	+	+
Capacity	+	+	+	+	-
Roles	+	-	+	+	-
Schedules	+	-	+	+	-
Cost of Usage	+	-	+	+	+
Multiple Roles	-	-	-	+	-
Duration	+	+	+	+	+
Warm-up Period	-	-	-	-	-
Replications	-	-	+	?	+
Confidence Intervals	-	-	-	-	-

Table 7.2: Updated evaluation of simulation tools using essential criteria

7.1.3 Updated Evaluation of the Tools

Now that we have fully evaluated Proter against Bonita and BIMP using a dataset designed to test all of the criteria identified in chapter 4, we can update the evaluation table from before to show the new capabilities of Proter (highlighted gray) alongside the other tools as shown in Table 7.2.

This table is slightly deceiving, as we have added more than it shows: BPMN support and the arrival process are very important additions yet these are not criteria found in the literature. Furthermore some of the criteria are actually slightly misleading, such as “Multiple Roles”. In Proter a task can have multiple resources but resources do not have roles, while BIMP only allows tasks to have a single resource, but since a resource has capacity it is considered to be a “role” by the surveys we have reviewed, yet in many ways a BIMP resource is essentially the same as a Proter resource.

7.2 Extending the Simulator Criteria

We have been using the list simulation criteria which we identified in Chapter 4 throughout this thesis for determining which features were missing in Proter up to creating example simulation scenarios to compare Proter with other tools. On multiple occasions

we have commented that this list, based on the evaluation of tools in the existing literature, is missing some criteria which we believe to be important. As such we suggest that this list of important functional criteria should be extended with the following:

1. **BPMN Model Support** - Throughout the literature reviewed in this thesis we see the recurring notion that the usage of BPMN in BPS greatly preferred over alternatives like Petri nets or proprietary solutions, for example in the work of Jansen-Vullers and Netjes [10], Peters et al. [25] and even in the Survival Guide [30]. Despite not being one of the criteria we identified from the literature, we also decided that support for BPMN is essential enough that it needed to be added to Proter. Due to this, and also with the knowledge that support for BPMN models is one of the first things that any review or tool survey mentions about a simulator, we believe it that this should clearly be included it in the list of essential criteria.

2. **Multiple Resources** - As we observed, the current "Multiple Roles" criterion is ambiguous. Bonita and Proter both allow tasks to require multiple resources, however since they do not support roles this criterion is not met. From the list of criteria it might seem that multiple resources are not a beneficial capability unless the tool also has resource roles, but this is false. Allowing multiple resources per task, regardless of whether roles are supported, can greatly improve the expressive potential of the model and therefore the quality of the simulation, and this is a big distinguishing factor between simulators which is why we propose that it belongs on the list.

3. **Prioritised Task Allocation** - This refers to the prioritised assignment of tasks to resources. In Proter, this is achieved through prioritised scheduling, and as we have shown in Chapter 3, the consideration of tasks priorities has a tangible effect on the simulation. Some simulators may have task priorities but still use first-come-first-served scheduling, and to our knowledge Proter is the only BPS simulator that has support for prioritised scheduling of tasks with multiple resources with a controllable degree of foresight such that lower priority tasks do not block or delay higher priority tasks.

   Prioritised scheduling is an approach to scheduling which is different to solutions such as the typical first-come-first-served strategy used in other simulators, but it is an addition and not a replacement. With prioritised scheduling, all of the same FIFO scenarios can be simulated simply by making all task priorities the same, but it also enables the creation of models which better reflect the real world and human behaviour by incorporating varying priorities. Given that this is an addition which enables the creation of better and more descriptive models for simulation, we believe this is an important addition to the simulation criteria.

The challenge in establishing and expanding on this list stems from the fact that there is no standard set of such simulator capabilities in the literature. We propose that the list we identified and expanded on in this thesis could be considered as a standardised list of essential BPS simulation capabilities. This list could be used in tool surveys to evaluate other business process simulators, and as a reference for the criteria which would be met by a theoretical ideal simulator, just as we have used it in this project.

Chapter 8

Conclusion

Business process simulation is a very powerful and widely used BPM technique, and simulations can benefit greatly from added expressive power which enables them to more closely model the real world. Our simulator, Proter, is designed and focused around a priority-centric approach to discrete event simulation, which enables workflows to be simulated in a more lifelike way. We believe this is a great and unique asset that could benefit many BPM professionals and researches. However, Proter was missing some critical functionality which could prevent or discourage its usage.

We set out to implement some essential features into Proter to meet the expectations of a business process simulator, and this thesis shows how we have identified and implemented them. We were also able to compare Proter to other business process simulators thanks to these additions, and show that Proter can execute the same simulation scenarios (modelled after the simulation criteria we identified) as other BPS tools.

This conclusion highlights the project outcomes, how this work fits in with existing literature, and discusses some shortcomings and potential improvements to our process.

8.1 Project Outcomes and Critical Evaluation

One of the main goals of this project was to improve Proter by adding some important features. We have succeeded in adding the following:

1. An Arrival Process
2. Arrival Distributions
3. Replications
4. Support for basic BPMN models
   (a) All standard BPMN activities- User, Script, Service, and Abstract tasks
   (b) Exclusive Gateways
   (c) Parallel Gateways

These additions enable Proter to simulate the same scenarios as other simulators, which we have successfully shown in our evaluation. The additional compatibility with the industry standard BPMN models also makes Proter more accessible and potentially useful to a wider audience.

These new features will not go to waste. Some of these, such as the arrival process and distributions, have already been merged onto the main release of Proter [1] and they will undoubtedly be used in future projects.

Unfortunately we were unable to implement all of the features that we had hopped. Warm-up periods and confidence intervals were also identified as being important criteria and a start was made in their implementation, but sadly we were not able to complete these features due to time constraints which resulted from our struggles with jBPM and Bonita. If we could anticipate the difficulties we encountered we could have found sufficient time to complete these features too.

The process of implementing BPMN support was also very challenging. Initially we hopped that the entire notation could be supported by using an external BPMN engine. However, as discussed in 6.2, this did not work out and we had to change plans. Many weeks were wasted on trying to get external libraries to work for our purposes, and this had a severe impact on other parts of the project. Ultimately we decided to parse BPMN ourselves, but we could only support the basic components given the limited time. Had we started with this approach, we could have added support for far more of the BPMN constructs.

The features of each simulator which we investigated were unclear in the beginning, and only after thorough investigation and having worked with them for a longer period of time did their unique differences and behaviour become apparent. At this point it was too late in the process to replicate their behaviour in Proter, which is why we have some different solutions to the same features, for example using a time limit in Proter instead of an instance count. These differences also made the process of evaluation more challenging, for example due to Bonita's strange injection periods and sampling intervals, and if we knew this earlier we could have tried to use different tools in our comparison.

Identifying and extending the list of critical simulation criteria was another goal of this thesis. We have successfully compiled a set of the criteria which appears most prominently throughout the literature, and added new criteria which we believe to be important. No other work attempts to identify and justify a standard set of such criteria, and so we propose that our identified list could fill this role. This list could serve as a guideline for creating the ideal business process simulator, and it could be used in future work for comparing and evaluating BPS tools.

As part of this project we also identified and discussed some very interesting differences in the way that the Bonita, BIMP, and Proter simulators work. As part of the evaluation process we also created a set of detailed simulation examples, which could be used in future tool surveys to evaluate other business process simulators.

[1] Main branch of Proter: `https://github.com/workflowfm/proter`

8.2 Future Work

Given our proposed standard for BPS simulator criteria, more discussion should be had around this topic. There is potential to propose other essential criteria which we did not discover in our investigation, or for critique of our choices, and in general we would like to see more effort in the direction of solidifying some sort of standard to which simulation tools can be developed and evaluated.

Proter can be further developed to implement the remaining missing criteria. The support for BPMN models is also incomplete, and future work could return to this implementation to add the missing constructs. This is not entirely trivial, since complex components like events and messages would require research to understand how they should be simulated.

Now that Proter has been improved with new features, future work could also apply Proter in new industry settings which can utilise the new functionality and further demonstrate the benefit of prioritised models in real applications.

8.2.1 Plan for Minf Project Part 2

In part 2 of this project we will turn our attention to the resource perspective. Now that Proter supports the essential simulation criteria, we can further improve it with the addition of features such as resource roles, capacity, multiple roles per resource, and schedules. These features would open up Proter to even more possible applications and use cases, and combined with its priority-centric approach it would provide an interesting perspective into the world of job shop scheduling and optimisation.

Combining multiple resources per task with resource roles and capacities and priority-based scheduling at the same time is a worthwhile challenge. The resulting environment would make the job of the scheduler much more complex since it would have to choose between multiple resources with various roles in a way that prioritises high-priority tasks. An exciting prospect is that this could possibly be framed as a more general optimisation problem which opens the doors for research involving possible machine learning and metaheuristic solutions even further down the line.

The baseline goals for part 2 of the project will be to:

1. Implement resource roles and capacity
2. Implement multiple roles per resource
3. Implement resource schedules
4. Provide basic scheduler implementations to support these additions (for example a greedy scheduler)
5. Provide an interface to the prioritised scheduling problem in Proter as an optimisation problem.

Bibliography

[1] Madis Abel. Lightning fast business process simulator. *Master's thesis. Institute of Computer Science, University of Tartu*, 2011.

[2] Cristina Adriana Alexandru, Daniel Clutterbuck, Petros Papapanagiotou, Jacques D Fleuriot, and Areti Manataki. A step towards the standardisation of hiv care practices. In *HEALTHINF*, pages 457–462, 2017.

[3] Rockwell Automation. Arena simulation software. `https://www.arenasimulation.com/`. Retrieved 9 April 2021.

[4] Gianfranco Balbo. Introduction to generalized stochastic petri nets. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 83–131. Springer, 2007.

[5] Bizagi. Simulation in bizagi. `https://help.bizagi.com/bpm-suite/en/index.html?simulation_in_bizagi.htm`. Retrieved 9 April 2021.

[6] Bonitasoft. Bonitasoft. `https://www.bonitasoft.com/`. Retrieved 9 April 2021.

[7] BOC Group. Adonis process simulation. `https://knowledge.boc-group.com/en/module/adonis-process-simulation/`. Retrieved 10 April 2021.

[8] KIE Group. jbpm. `https://www.jbpm.org/`. Retrieved 9 April 2021.

[9] Michael Hammer. What is business process management? In *Handbook on business process management 1*, pages 3–16. Springer, 2015.

[10] Monique Jansen-Vullers and Mariska Netjes. Business process simulation–a tool survey. In *Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, Denmark*, volume 38, 2006.

[11] Kurt Jensen. Coloured petri nets. In *Petri nets: central models and their properties*, pages 248–299. Springer, 1987.

[12] Kurt Jensen, Lars Michael Kristensen, and Lisa Wells. Coloured petri nets and cpn tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer*, 9(3-4):213–254, 2007.

[13] W David Kelton. *Simulation with ARENA*. McGraw-hill, 2002.

[14] Averill M Law, W David Kelton, and W David Kelton. *Simulation modeling and analysis*, volume 3. McGraw-Hill New York, 2000.

[15] Camunda Ltd. Camunda. `https://camunda.com/`. Retrieved 9 April 2021.

[16] Visual Paradigm International Ltd. Visual paradigm. `https://www.visual-paradigm.com/`. Retrieved 10 April 2021.

[17] M Ajmone Marsan, Gianfranco Balbo, Gianni Conte, Susanna Donatelli, and Giuliana Franceschinis. Modelling with generalized stochastic petri nets. *ACM SIGMETRICS performance evaluation review*, 26(2):2, 1998.

[18] OMG. Business process modeling notation (bpmn) (2011). `https://www.omg.org/spec/BPMN/2.0/PDF`. Retrieved 23 October 2020.

[19] MABEL OÜ. Bimp simulator. `https://bimp.cs.ut.ee/simulator/`. Retrieved 9 April 2021.

[20] Nathaniel Palmer. What is bpm? `https://bpm.com/what-is-bpm`. Retrieved 23 October 2020.

[21] Petros Papapanagiotou and Jacques Fleuriot. Workflowfm: A logic-based framework for formal process specification and composition. In *International Conference on Automated Deduction*, pages 357–370. Springer, 2017.

[22] Petros Papapanagiotou, James Vaughan, Filip Smola, and Jacques Fleuriot. A real-world case study of process and data driven predictive analytics for manufacturing workflows. In *Proceedings of the 54th Hawaii International Conference on System Sciences*. Hawaii International Conference on System Sciences, 2021. To appear.

[23] José Luís Pereira and António Paulo Freitas. Simulation of bpmn process models: Current bpm tools capabilities. In *New Advances in Information Systems and Technologies*, pages 557–566. Springer, 2016.

[24] Sander PF Peters, Remco M Dijkman, and Paul WPJ Grefen. Advanced simulation of resource constructs in business process models. In *International Conference on Business Process Management*, pages 159–175. Springer, 2018.

[25] Sander PF Peters, Remco M Dijkman, and Paul WPJ Grefen. Advanced simulation of resource constructs in business process models. In *International Conference on Business Process Management*, pages 159–175. Springer, 2018.

[26] WorkflowFM Petros Papapanagiotou. Proter. `http://docs.workflowfm.com/proter/`. Retrieved 9 April 2021.

[27] Nick Russell, Wil MP van der Aalst, Arthur HM Ter Hofstede, and David Edmond. Workflow resource patterns: Identification, representation and tool support. In *International Conference on Advanced Information Systems Engineering*, pages 216–232. Springer, 2005.

[28] Spray. Spray-json library. `https://github.com/spray/spray-json`. Retrieved 9 April 2021.

[29] CPN Tools. Cpn tools. `https://cpntools.org/`. Retrieved 9 April 2021.

[30] Wil MP Van Der Aalst. Business process simulation survival guide. In *Handbook on Business Process Management 1*, pages 337–370. Springer, 2015.

[31] Wil MP Van der Aalst, Joyce Nakatumba, Anne Rozinat, and Nick Russell. Business process simulation. In *Handbook on Business Process Management 1*, pages 313–338. Springer, 2010.

[32] Wil MP van Der Aalst, Arthur HM Ter Hofstede, Bartek Kiepuszewski, and Alistair P Barros. Workflow patterns. *Distributed and parallel databases*, 14(1):5–51, 2003.

[33] Wil MP van Der Aalst, Arthur HM Ter Hofstede, Bartek Kiepuszewski, and Alistair P Barros. Workflow patterns. *Distributed and parallel databases*, 14(1):5–51, 2003.

[34] Wil MP Van Der Aalst, Arthur HM Ter Hofstede, and Mathias Weske. Business process management: A survey. In *International conference on business process management*, pages 1–12. Springer, 2003.

[35] Petia Wohed, Wil MP van der Aalst, Marlon Dumas, Arthur HM ter Hofstede, and Nick Russell. On the suitability of bpmn for business process modelling. In *International conference on business process management*, pages 161–176. Springer, 2006.

[36] Michael Zur Muehlen and Jan Recker. How much language is enough? theoretical and practical use of the business process modeling notation. In *Seminal Contributions to Information Systems Engineering*, pages 429–443. Springer, 2013.

Appendix A

Evaluation Results

A.1 Evaluation Examples Dataset Reference

1. (Basic Flow) Sequence, all one resource
2. (Basic Flow) Parallelism, all one resource
3. (Basic Flow) Parallelism, all different resource
4. (Basic Flow) Parallelism, one conflicting resource
5. (Basic Flow) Branching, all one resource
6. (Basic Flow) Branching, all different resource
7. (Basic Flow) Branching, one conflicting resource, tight timing
8. Long Parallelism, pairwise conflicting resources - each resource used by two parallel tasks
9. Long Parallelism, Staggered resources - each parallel task uses different resources
10. Long Parallelism, varying durations, semi-conflicting resources (staggered but time overlap)
11. Long Parallelism, same as (8) but short time (perfect time due to resource alignment)
12. Long Parallelism, same as (9) but short time (should be perfect amount)
13. Simple loop - one task
14. Long loop - multiple tasks
15. Large branching of two parallelisms - large BPMN example
16. Large parallelism of two branchings - large BPMN example
17. “validate customer” - BPMN from Camunda examples
18. “invoice approval” - BPMN from Camunda examples
19. “calculate rating” - BPMN from Camunda examples
20. “parallel” - BPMN from Scylla examples
21. Basic model, random arrival rate
22. Basic model, random task durations (uniform)
23. Basic model, random arrival rate + random task durations

A.2 Results

The results table has been split into three in order to fit on the page.

	Proter					
	Resource Utilisation (%)				Instances	
Example	r1	r2	r3	r4	Average Waiting Time	Average Execution Time
1	75	0	0	0	0	15
2	100	0	0	0	5	20
3	25	25	25	25	0	15
4	25	50	25	0	5	20
5	75	0	0	0	0	15
6	25	13.5	11.5	25	0	15
7	33.33	33.33	33.33	0	0	15
8	33.33	33.33	33.33	0	5	20
9	33.33	33.33	33.33	0	0	15
10	16.67	33.33	36.67	0	4	16
11	66.67	66.67	66.33	0	5	20
12	66.67	66.67	66.67	0	0	15
13	63.33	0	0	0	1.7	11.2
14	70.33	70.33	70	0	4.15	35.75
15	25	25	25	0	0	25
16	19.5	43.62	18.88	18	0	25
17	50	25	0	0	0	15
18	17	83.67	0	0	5.4	21.2
19	66.67	36	0	0	2.3	17.7
20	20	18.1	18.1	18.1	0	19.05
21	33	32.67	33	0	2.11	10.99
22	93	44.93	0	0	8.63	17.97
23	83.53	42.73	0	0	69.37	49.43

Figure A.1: Evaluation results for Proter

	Bonita					
	Total Resource Utilisation (%)				Instances	
Example	r1	r2	r3	r4	Average Waiting Time	Average Execution Time
1	75.7	0	0	0	0	15.1
2	99.3	0	0	0	0	20.15
3	25.4	25.4	25.4	25.4	0	15
4	25.3	50.6	25.3	0	0	20
5	75.7	0	0	0	0	15.1
6	25.4	0	25.4	25.4	0	15
7	33.9	33.9	33.9	0	0	15
8	33.7	33.7	33.7	0	0	20.05
9	33.7	33.7	33.7	0	0	15.05
10	16.8	33.7	37.1	0	0	16.07
11	66.4	66.4	66.4	0	0	20
12	66.4	66.4	66.4	0	0	15.05
13	33.6	0	0	0	0	5
14	DNF	DNF	DNF	DNF	DNF	DNF
15	25.1	25.1	25.1	0	0	25
16	25	37.6	25	12.5	0	25.1
17	50.7	25.4	0	0	0	15
18	0.33	0.99	0	0	0	27.5
19	DNF	DNF	DNF	DNF	DNF	DNF
20	DNF	DNF	DNF	DNF	DNF	DNF
21	33.4	33.4	33.4	0	0	10
22	66.7	33.3	0	0	0	10.05
23	66.7	33.3	0	0	0	10.05

Figure A.2: Evaluation results for Bonita

	BIMP					
	Resource Utilisation (%)				Instances	
Example	r1	r2	r3	r4	Average Waiting Time	Average Execution Time
1	75.19	0	0	0	0.5	15
2	100	0	0	0	5.5	20
3	25.06	25.06	25.06	25.06	0.5	15
4	25	50	25	0	5.5	20
5	75.19	0	0	0	0.5	15
6	25.06	12.53	12.53	25.06	0.5	15
7	33.33	33.33	33.33	0	0.5	15
8	33.44	33.44	33.44	0	5.5	20
9	33.5	33.5	33.5	0	0.5	15
10	16.74	33.49	36.84	0	4.5	16
11	66.45	66.45	66.45	0	5.5	20
12	66.67	66.67	66.67	0	0.5	15
13	64.21	0	0	0	1.58	11.1
14	58.1	58.1	58.1	0	2.26	30.6
15	25.09	25.09	25.09	0	0.5	25
16	18.19	43.41	19.32	19.45	0.5	25
17	50.13	25.06	0	0	0.5	15
18	15.89	82.12	0	0	2.9	19.6
19	66.45	35.22	0	0	1.28	19.2
20	20.1	18.99	18.99	18.99	0.5	19.5
21	34.31	34.31	34.31	0	2.24	10.8
22	99.37	50.37	0	0	23.56	23.8
23	92.48	43.72	0	0	185.81	108

Figure A.3: Evaluation results for BIMP