Experimental Comparison of Open Source Discrete-Event Simulation Frameworks

Oskar Skak Kristiansen, Ulrik Sandberg, Casper Hansen, Morten Skovgaard Jensen, Jonas Friederich(✉), and Sanja Lazarova-Molnar

Mærsk Mc-Kinney Møller Institute, University of Southern Denmark, Campusvej 55, 5000 Odense, Denmark
jofr@mmmi.sdu.dk

Abstract. Simulation is a growing and very relevant field today. With many application domains and increasing computational power, simulations are solving an increasing number of real-world problems in a safe and efficient manner. The aim of this study is to make an experimental comparison among popular open-source discrete-event simulation frameworks, selected through a thorough survey. The experimental comparison considers numerous parameters, the most important of which were studied via an operational experiment of performance, as well as a survey of surface level characteristics and usability. We initially surveyed 50 frameworks for high-level parameters relating to corresponding implementations, filtered by the generality of domain, license, popularity, etc. As a result, we shortlisted five frameworks, which we then performance-tested in an operational experiment. We discovered significant performance differences under the given loads. We also completed a usability survey to provide a holistic impression of the frameworks, further identifying key differences.

Keywords: Discrete-event simulation · Open source · Performance comparison

1 Introduction

Simulation is a growing and very relevant field today. With many application domains and increasing computational power, simulations are solving an increasing number of real-world problems in a safe and efficient manner [1, 2]. Simulation is often used to conduct experiments where a real-world model would be either impractical or even impossible to make, saving on both cost and/or time. This enables analysts, engineers and stakeholders to communicate, verify and understand concepts and ideas concerning complex systems [3].

In the current literature, we have not identified a comprehensive comparison of open-source simulation frameworks in terms of the parameters that we investigated, i.e., performance, lines of code (LOC) for minimal implementation, community engagement, etc. [4]–[8]. In this article, we share our findings from investigating and comparing various open-source libraries available for discrete-event simulation (DES). Furthermore, we use a classical simulation model case study to experimentally perform the comparison.

D. Jiang and H. Song (Eds.): SIMUtools 2021, LNICST 424, pp. 315–330, 2022.
https://doi.org/10.1007/978-3-030-97124-3_24

While no similar comparison is available, there are numerous comprehensive studies comparing simulation frameworks in different specific or even more general contexts [6, 8, 9]. With this, we add to the current literature a quantitative comparison of several state-of-the-art open-source simulation frameworks in a more experimental and practical manner.

Being a relatively large field of study, it is obvious that choosing the right simulation tool for the job is one of the initial important decisions with high impact on the results. This is emphasized by performance as well as functionality required by the researcher often being a limiting factor for selection of tools. As many libraries exist, choosing the right one is not trivial.

As noted, the aim of this study is to compare among popular open-source simulation libraries in an experimental way. The purpose of this comparison is to provide researchers that need to use open-source DES frameworks, a guidance for a more informed choice. The study will seek to answer the following research questions: 1) What open-source modelling and simulation libraries exist? 2) How do these open-source libraries compare on specific parameters related to the quality attributes described within the field of software engineering and related categorical differences?, and 3) Which frameworks perform the best during run-time in regards to performance and related quantitative measures? In answering these questions, we aim to provide simulation practitioners with insight in the open-source modelling and simulation frameworks included in this study.

The paper is structured as follows. In Sect. 2, we examine the existing literature related to open-source DES frameworks. We present the overall research methodology involved in Sect. 3. This includes the search process that we used to identify articles and frameworks relevant to the context of open-source simulation frameworks. In the same section we also present the primary research method used in this project. In Sect. 4 we present and discuss the results of the operational experiment and the usability survey. We discuss whether a recommendation can be made from the investigated frameworks in Sect. 5, and lastly conclude in Sect. 6.

2 Related Work

The field of simulation is, while growing, already widely used throughout many disciplines. As a consequence, many studies comparing existing frameworks in different, and often quite specific contexts have been done.

To identify articles, similar in nature to our point of interest, we utilized scholarly database querying, which we further detail in Sect. 3.2. While no exact duplication in topic and context could be found, many related works exist. This includes works such as that of Franceschini et al., which is quite similar in nature to this project, except for its inclusion of proprietary simulation frameworks whereas we limit our investigation to open-source frameworks as well as the parameters included in the comparison, such as the ease of use comparison as well as some performance measures [5]. Other works include those by Majid et al. that has a similar methodology and purpose, with the

significant difference of comparing frameworks on the accuracy parameter instead of performance, as well as the work by Dagkakis and Heavy, which focuses on operations research, and Knyazkov and Kovalchuk, which while similar in methodology, limits its scope to interactive virtual environments [4, 6, 7]. Further, analyses of single frameworks are quite exhaustive, including Göbel et. al., which focuses on DESMO-J, and Barlas and Heavy, whose focus is on the Knowledge Extraction Tool, all of which are quite useful in informing about the available frameworks themselves, but does not constitute the comparison between frameworks this study attempts to lay out [8, 10].

As implied above, while much published research in the field of modelling and simulation exists, they all differ in their primary focus, such as comparing frameworks on another criteria than performance, or in their criteria for inclusion in the comparison, such as not focusing exclusively on open source frameworks. This is noted in the aforementioned Dagkakis and Heavey journal, wherein they suggest future work should include a weighted comparison of open source DES frameworks (that is, the objective of this paper) [4]. Beyond the differing focus and chosen parameters of related works, another important aspect of these works is the time period in which they were conducted. Per the nature of the ever-evolving open source landscape, particularly in such an active field, new analysis of frameworks is needed intermittently, as any attempt of illuminating the state of the art landscape will quickly be partially outdated if not entirely deprecated. Lastly, we should note the field of study is simply too big as for any project to include all related existing research in its analysis. This is not of major concern in regard to the novelty of this project, as the methodology is quite specific, and the emphasis on the various parameters is almost sure to be unique. Further, although the environment is very controlled, different algorithmic comparisons will yield vastly different results in regard to performance of the selected frameworks, meaning the resulting comparison is sure to contribute to existing material.

3 Research Methodology

In this section we describe the research methodology for our survey, including the experiment but also the formulation of the evaluation criteria and the subsequent filtering of the numerous simulation frameworks under initial consideration.

3.1 Search Process

For the purpose of identifying related works and potential gaps in the existing literature, we utilized slightly broad search terms in the Web of Science academic research database. We narrowed down the search through additional constraints, including time, relevance, number of citations, etc., to identify articles relating to the relatively broad problem definition of this study. From these articles, we identified potential candidates for the technical comparison of open source DES frameworks through either direct mention in the given articles, or through references. With this methodology, we extracted 50 candidates for potential testing. We filtered out the candidates that are no longer actively supported in combination with additional filter: "LastUpdated $\geq 2018 - 01 - 01$". The result was the 13 frameworks shown in Table 1.

Table 1. Framework classification.

Name	Year	Domain	OS	Language	Ext	Lic	Vis	Engagement
SimPy	20	3	3	Python	1	3	–	26 (26s + 0f)
SimJulia	19	3	3	Julia	1	3	–	138 (22s + 106f)
OMNeT	19	3	3	C++	2	2	3D	277 (226s + 51f)
JaamSim	20	3	3	Java	2	3	3D	100 (77s + 33f)
NS-3	20	1	3	C++	2	2	3D	578 (202s + 376f)
Facsimile	20	2	3	Scala	1	3	–	26 (20s + 16f)
JavaSim	20	3	3	Java	1	3	–	36 (20s + 16f)
J-Sim	20	3	3	Java	1	3	2D	2 (1s + 1f)
Root-Sim	19	3	1	C	1	2	–	28 (13s + 15f)
TerraME	20	1	3	Lua	1	3	2D	41 (27s + 14f)
SSJ	18	3	3	Java	1	3	–	101 (68s + 33f)
VLE	19	3	3	C++	1	2	3D	34 (14s + 20f)
ADevs	20	3	2	Java	1	3	–	11 (4s + 7f)

The reasoning for excluding frameworks not updated before the specified date is that software frameworks need to update often, if not for functional reasons, then at least as a response to the ever-changing meta environment of the different operating systems, protocols of communication, etc. [11]. After the filtering on dates, we researched the remaining 13 frameworks and identified the most important characteristics, to give a more holistic impression of the overall collection. We summarize our findings in Table 1. For the domain ('Dom'), we gave a 1–3 score with 1 being very domain specific, 2 allowing generic use, and 3 decidedly generic in its domain. The OS columns refers to how many of the three most common operating systems (Microsoft, Mac OS and Linux-based) the frameworks are directly integrated into (meaning the frameworks can function without extensive custom configuration, usage of virtual machines, etc.). For the extension category ('Ext'), we denoted frameworks with only manual custom extension with 1, and frameworks with specific mechanisms for extension, such as offering virtual extension methods on existing modules which allows for modification without changing the underlying architecture or offering convenient ways of defining new options, data structures, etc., with 2. With license ('Lic') we denote frameworks with 1 if no distribution is allowed, 2 if only non-commercial distribution allowed, and 3 if the license is completely open. The Engagement category shows the GitHub star (s) and forking (f) score of the framework repositories.

To arrive at the five chosen frameworks, the following filters were used: (DomainScore $==3) \wedge$ (LicenseScore $==3) \wedge$ (Max(Engagement)). There are several reasons for the choice of these filters. First, all 13 frameworks are recently updated and have decent possibilities in terms of development environments This means the filtering must be done on other criteria. Parameters, such as a graphical user interface (GUI) environment for development, documentation and integrated extension possibilities cannot be said to be objective measures of suitability, since their importance is almost entirely dependent on the context. Domain (as defined in the summation above), however, is obviously quite important in this context as the aim is to analyse generic frameworks. Further, licensing is of vital importance, as whatever the technical benefits of a framework might be, if the licensing is not permissive enough the framework is simply not usable in many cases. In the last filter the frameworks with the greatest level of engagement are chosen. From the sequential filtering, five frameworks were chosen to be included in the experiment. Each framework will be introduced below.

SimPy is based on standard Python, and can be described as a domain specific library. It is specifically designed for DES, but can obviously be extended to include continuous simulation through custom extension in Python [12]. It is distributed under the open MIT license, meaning all modifications and extensions are allowed, and maintains an active and widely engaged community.

SimJulia differs from SimPy in that continuous event processing is directly integrated, along with discrete event processing which is what we are interested in, in this context. It is technically quite similar to SimPy, in that the API modelling imitates that of SimPy [13]. Its documentation and community engagement are extensive, and the licensing is identical to that of SimPy.

JaamSim differs from SimPy and SimJulia in its method of simulation implementation, in that developers can use its IDE (Integrated Development Environment) and integrated graphical input functionality. Beyond that, it is quite similar in supported functionality and like SimPy, focuses directly on DES [14]. The functionality offered in the IDE is quite extensive, but potential users should still realize that many third-party opportunities are perhaps better integrated in the CLI (Command Line Interface) style development frameworks described above.

SSJ (Stochastic Simulation in Java) is, as the name suggests, a Java-based simulation framework, whose primary focus is on DES [15]. As such, SSJ includes the expected tools and methods for developing standard DES simulations. SSJ supports both continuous and hybrid simulation, and allows developers to implement models in a variety of general purpose languages, as well as simulation specific languages [15].

JavaSim is described as an ‘object-oriented, discrete event simulation toolkit’ [16], directly integrated with the general purpose Java language despite the specific paradigm, with a focus on flexibility, extensibility and efficiency. The specific simulation paradigm supported by JavaSim is ‘continuous time-discrete event’, which means it includes the expected tools relating to the context of this paper and allowed for relatively straightforward implementation of the chosen model.

3.2 Method Formulation

The primary research method in this study is experimental, specifically, a controlled experiment within the domain of Software Engineering, as described in "A Survey of Controlled Experiments in Software Engineering" [17]. The definition of a controlled experiment in this context is operational, and relates quite closely to the quasi-experimental method in classic scientific terminology, in that it involved conducting software engineering tasks to observe and compare processes, methods, etc. [17]. This definition and related procedures follows the stated purpose more closely than the different classical definitions, as there obviously is no need to introduce control groups or randomization in order to assess the chosen parameters [18], as the experiment can be directly controlled in a relatively precise manner. The experiment will be used to test parameters related to the performance of the included frameworks, specifically computational time under given loads as well as CPU usage, which are of significant importance within the field of simulation, given its resource intensive nature.

Beyond the primary method of experimentation, an arguably significant part of the comparison involves the characteristics described in Sect. 3.1, as well as the unstructured usability survey described in Sect. 3.5. As is evident, the procedure of this study will be exploratory and descriptive, as is commonplace in this context [19].

3.3 Experimental Simulation Model

We designed the simulation model to contain elements that are typical for discrete stochastic models. The simulation model used for the experiment emulates a street food like scenario, with three important aspects, namely customers are able to order multiple times, there is a limited amount of customers that can be in the street food at once and they pay before they leave.

In the simulation, agents "arrive" at the street food according to an exponential distribution. Following arrival, customers queue for a table, after which they decide to either order food or drinks immediately. If they decide to order food, they will distribute over three different food stands, namely burger, pizza or Chinese with an equal probability. For each food stand there is an associated probability distribution with both waiting for the food, as well as eating it afterwards. If a customer chooses to order a drink instead of food, he/she will immediately queue at the drink stand. Once served, he/she will decide if he/she also wants to order food. If he/she again decides to order food, he/she will proceed through the food order flow described earlier. When a customer is done eating or drinking, he/she has a final decision, he/she can either repeat the entire flow and thus order food/drinks again or leave the street food through a checkout process.

The model is formulated to be simple enough to not introduce errors or disputes in implementation across different frameworks, while still containing several of the elements which are ordinarily used in DES, such as limited relative resource usage, concurrency and generation of values from commonly used distributions.

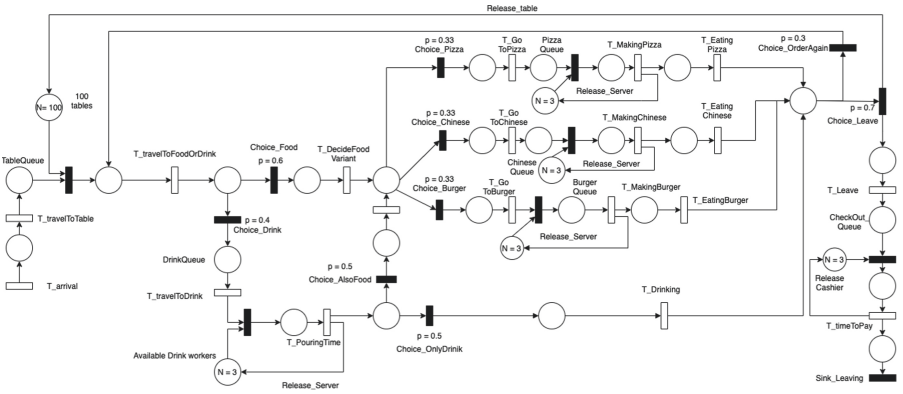

Fig. 1. Stochastic Petri Net of the experimental simulation model.

3.4 Framework Testing

We use the primary exploratory experiment, as described above, to compare the performance of the selected simulation frameworks during what could be described as normal use within the context of DES. That is to say, the model described above means to mirror a typical relatively low level task, albeit likely less intensive. However, since there is a variety of factors that can potentially influence the performance parameter, such as spikes in resource usage of background tasks or scheduled processes occurring, even within the relatively controlled environment, measures were taken to minimize influences external to the experiment itself. Since the complexity of the tested frameworks is of a relatively high degree, it is not necessarily the case that a fair comparison can be made with derived generic conclusions if the input to the simulation model is the same across all tests. Reasons for this include, as an example, the benefits obtained from overhead in a framework potentially being helpful when processing a resource intensive task, but has a negative trade-off in terms of comparison if the load is small, since such optimizations potentially would not outweigh their cost if the runtime is small. To avoid judging on a biased foundation, the simulation model described above is ran with an increasing input load from 25,000 to 500,000 customers (n), increasing by 25,000 per interval. Further, to reduce the variation in results as a consequence of environmental factors (such as the various background processes, services, etc.) each input interval is replicated ten times. To validate the summarized results, each individual run duration was inspected, ensuring no outliers would significantly diverge the summed results. The CPU usage percentage as well as computational time for each interval was recorded in order to properly analyse the performance of each framework. The tests were ran in a stable and as close to identical environment as possible to make the results quantitatively comparable. Each test can be seen in the projects repository [20], along with the implementation of the model in each of the tested frameworks.

3.5 Configuration Details

The hardware configuration of our test setup was as follows:

- CPU: Intel Core™ i9-9880H CPU @ 2.30 GHz
- GPU: AMD Radeon Pro 5500M 4 GB
- RAM: 8 * 2 GB DDR4 @ 2667 MHz

The versioning of the included frameworks is shown below in Table 2.

Table 2. Versions of included frameworks

Name	Version	Language Version
Javasim	2.3	15.0.2 (java)
JaamSim	2021.01	15.0.2 (java)
SSJ	3.3.1	15.0.2 (java)
Simply	4.0.1	2.7.16 (python)
SimJulia	1.2	15.0.2 (java)

3.6 Usability Comparison

The usability comparison was done in a relatively informal manner, since the primary focus of the applied research methodology was to test the performance and other naturally quantifiable aspects. We still feel, however, that even a subjective usability assessment is useful to include, since it can be helpful to potential users. While there are formalized quantifiable measures for usability, we found many of the methodologies presented were not fully applicable and did not necessarily give a fair picture given the experience gained in the sequential process of developing the model in each framework. We deemed it best to simply reflect on important parameters regarding usability and derive a 'score' which was relative to the other frameworks, and simulation frameworks in general. Specifically, each developer filled a simple survey asking them to judge the given parameters within an interval from one to five, and then generalize the results. The specific parameters are presented in Sect. 4.1 along with how each parameter scores.

4 Results

In this section we present and analyse the data gathered from the aforementioned tests. This includes the results of the performance tests, but also a more usability focused analysis of the utilization of the different frameworks based on the experience of working with the frameworks in this project. Lastly, we will discuss the results and explore aspects related to the observed differences in performance and usability.

4.1 Performance Comparison

As mentioned in Sect. 3.4, each framework was tested under increasing load in the interval from n = 25,000 to n = 500,000, with each interval having a range of 25,000, meaning 20 individual results per framework were gathered. Further, each interval was repeated ten times to ensure the results would not be overly biased by outliers. In Fig. 2, the averaged computational times are shown.

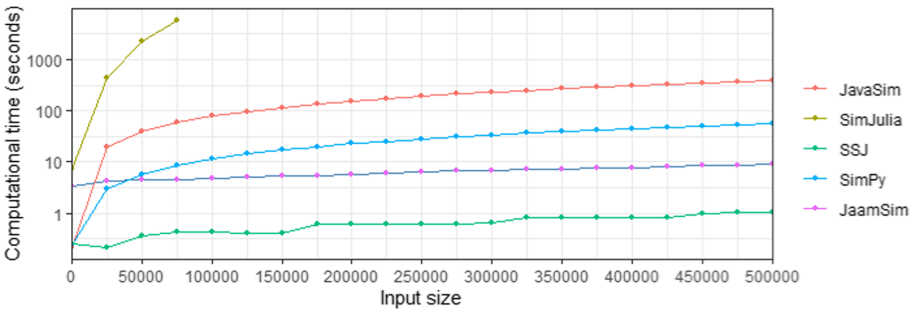

Fig. 2. Computational time for each framework given varying input size.

As is evident, the difference in performance under the given conditions and environment is very significant. Note, that due to time constraints and the very evident degradation in performance under increasing loads, the tests for SimJulia were halted after the tests with input size equal to 75,000. While JaamSim, SimPy, JavaSim and SSJ frameworks are somewhat comparable in performance, and follow a clearly evident linear progression under increasing load, SimJulia has a lot of variance between results internally, and is vastly slower under the recorded range of loads. Since SimJulia presents such a difference in both overall results and internal variance, further investigation of the framework was performed. Here, we focus on its difference to SimPy (a comparison with the other frameworks will be further explored below). As mentioned, SimJulia is written after the specification of the SimPy API. This means the internal processing is expressed syntactically very similar between SimJulia and SimPy. While this is convenient when attempting to perform functionally identical tests, it also means the possibility of human errors introduced in the test definition is quite low, since the given functionality needs only minimal translation to be functional in either environment. Further, to ensure the very significant difference and internal variance is not due to factors in the external environment, SimJulia's test was replicated numerous times in the environment, with similar results each time. While mistakes are always possible, the relative simplicity of the simulation model, along with the syntactic and functional similarity between SimJulia and SimPy, leads us to conclude that SimJulia likely simply performs far below the level of the other frameworks. This is further backed up by the large degree of popularity and extensiveness of the remaining frameworks relative to SimJulia. Simulation processing under a relatively large load is obviously quite complicated, and it seems SimJulia simply is not as fine-grained in its performance under increasing loads, as the other frameworks under consideration.

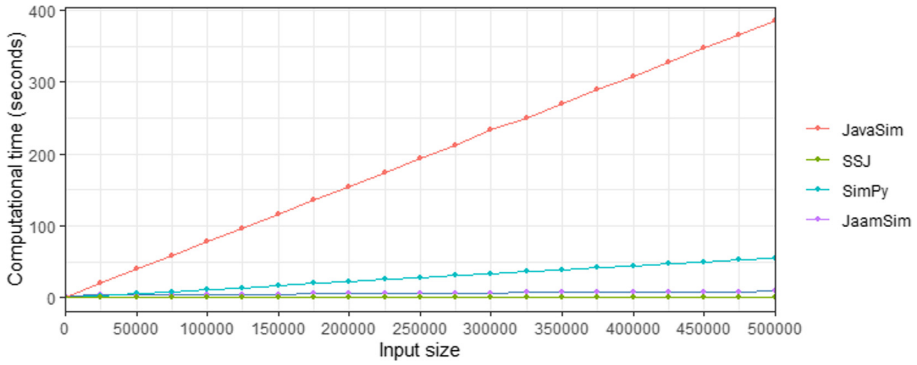

Fig. 3. Computational time for each framework given varying input size excluding SimJulia.

From Fig. 2, it is evident that the overall duration of processing across all input sizes is much longer in SimJulia, being hardly comparable to the remaining frameworks. For this reason, we will exclude SimJulia in the following discussion. In future work it could be interesting to investigate the reason for this drastic difference in performance. In Fig. 3 we show test results of the remaining frameworks.

As is evident, while all frameworks produce processing times which clearly have linear relationships to the loads, JaamSim and SSJ have a significantly smaller rate of growth. Practically, this means JaamSim and SSJ produce very similar processing times under the initial (and relatively light) load, but that the difference between the frameworks grows linearly, and JaamSim and SSJ outperforms SimPy and JavaSim increasingly significantly while the load increases. In future work, it could be interesting to explore if the exponential relationship between the graphs means SimPy actually has better performance with loads below the initial interval used here (that is, 25,000), as is loosely implied by the growing difference. Specifically, if the graph followed the implied relationship, it appears SimPy could outperform JaamSim in loads with input size slightly below the lower bounds in the experiment of 25,000.

In Fig. 4, the average CPU usages, in percentage, across the tested input intervals are shown. It is obvious from the graph, that JaamSim occupies a substantially greater percentage of the CPU for the lower input sizes, before gradually decreasing until it is around the level of the remaining frameworks. This is expected, given the nature of the JaamSim framework. While the remaining frameworks are integrated directly into general purpose programming languages, and could be described as a simulation library in the given languages, JaamSim is a standalone framework. One reason for the greater CPU usage could be attributed to the overhead of the standalone framework, even if it is compiled into an executable jar, similar to JavaSim and SSJ. This is supported by the development of the graph, since the overhead, if associated with the start-up of the framework, obviously would be a smaller part in the latter intervals, and the CPU usage, therefore, remains steady once the cost of the overhead is factored out by the larger overall cost of the framework running in steady use. In Fig. 5, we can see the CPU usage of JaamSim in the interval where input size equals 32,500, that is, the interval before the drop-off to a steady usage is found. Examining the graph, it is evident that a substantial

percentage of the CPU is utilized by JaamSim in its beginning stages of execution, which drops off, in this case, after about 18 s (note that the run depicted shows 10 replications of the simulation).

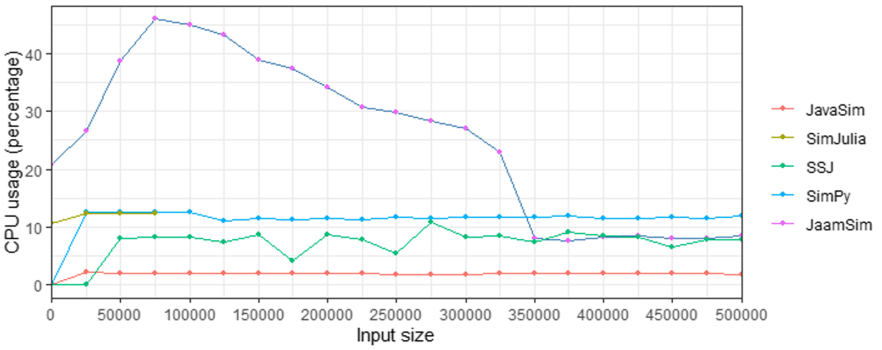

Fig. 4. Average CPU usage percentage across varying input sizes.

The trend identified in the CPU usage of JaamSim is not found in any of the CLI styled frameworks, which all remain relatively steady in their CPU usage throughout execution, despite the load. To conclude, it is evident that JaamSim is quite resource heavy in its beginning execution, which then tampers off during continuous use. Further JavaSim allocated substantially fewer resources than any other framework, which makes sense given its relatively slower execution across all tested intervals. SSJ and SimPy are quite similar in their CPU usage throughout the intervals, indicating there must be another reason for the significant difference in computational time needed to complete the simulations. Finally, SimJulia, for the few tested input sizes, is comparable to the other CLI styled frameworks in its CPU usage, meaning a lack of resource allocation is evidently not the reason for the reduced performance in comparison to the other frameworks.

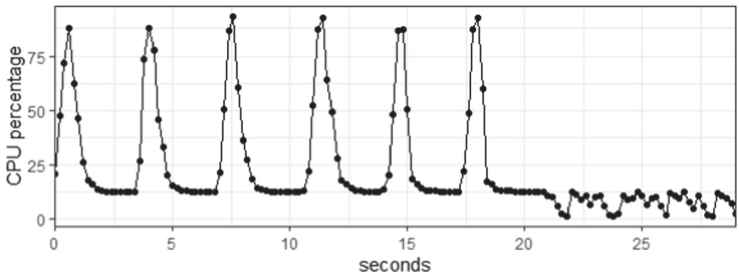

Fig. 5. JaamSim's CPU usage during runtime with n = 32,500.

To conclude, as is readily evident in the gathered results, there is a clear difference in performance between the frameworks, with SSJ outperforming the other frameworks in all tests within the given range. JaamSim is slightly behind SSJ on this parameter

but follow relatively closely. This difference grows at an exponential rate in comparison to JavaSim and SimPy, meaning JaamSim and SSJ performs relatively better under increasing loads. SimPy is only slightly behind in the initial intervals but is slower throughout the range. JavaSim has a linear development in computational time across the tested intervals, but is significantly slower than JaamSim, SSJ and SimPy. Lastly, SimJulia scores significantly lower in performance relative to the other frameworks under consideration, at all stages, and has a variance in performance not found in the other tested frameworks.

4.2 Usability Comparison

In this section, we present the results of the usability comparison.

JaamSim had various generic installers available, including 64/32-bit windows installers and compiled jar executables for all major OSs. JaamSim, whose input is configured through its GUI, is documented thoroughly, including start-up examples of different available basic models. We found JaamSim intuitive, given the GUI, albeit with relatively poor debugging opportunities. Documentation and community engagement seems to be very limited on the more specific issues.

SimPy does not have a dedicated installer, given its integration in standard Python. This means the installation is specific to the given users' context, such as IDE integration, pip, etc. As a non-GUI framework, the documentation felt extra critical in order to facilitate an efficient workflow. Fortunately, both core concepts of the framework as well as DES vocabulary had been maintained throughout the documentation. It also featured blogs, examples and start-up walk-through guides. Effectively, this meant that the chosen simulation model was implemented easily. Besides the documentation, another aspect which felt beneficial and greatly impacted the feeling of "ease of use" was that we were free to extend, modify and structure the code as it optimally fitted our needs.

We found SimJulia had a far greater barrier to entrance than the other frameworks. Like SimPy, SimJulia is CLI based. It was quite difficult to use, relative to other frameworks. Setting up the correct SimJulia environment proved to be very difficult. The official documentation was incomplete and deprecated. Navigating to the projects' GitHub page resulted in conflicting results, as it recommended an older version of Julia than the documentation. This change, however, did not solve the documented examples. Very little community guidance exists, and we had to look to the SimJulia repositories test suite for the particular versioning. Once up and running, the model was easier to implement, however this was more a tribute to the completeness of SimPy's documentation and the fact that SimJulia's had modelled their API after SimPy's implementation.

The initial installation and setup of JavaSim was quite straightforward and well documented. Further, while documentation could not be said to be extensive, it did include some examples to illustrate the most basic tools of DES included in the framework. These examples were not as fully illustrative as would be ideal, but they were a good starting point. The framework did require more than surface level knowledge of Java, since things like parallelism, which is obviously vital to performance in simulation, were mostly left up to the developer to manage.

SSJ, like the other CLI based frameworks, does not come with an integrated installer, but is available through most of the common package managers such as Maven, Gradle, etc. The documentation was extensive, easy to understand and included examples. Additionally, a lot of examples were also provided in the repository which gave intuitive examples to work with. An important aspect of SSJ was its emphasis on using the object oriented paradigm. When compared to the more scripting oriented languages, SSJ felt a lot more structured and making more complicated models in SSJ feels facilitated better in comparison to the remaining frameworks. In Table 3 the score of the chosen parameters of usability is summarized.

Table 3. Ease of use comparison.

	SimJulia	JaamSim	SimPy	SSJ	JavaSim
Startup documentation	1	5	4	4	3
Best practice examples	1	3	5	5	3
Extension options	3	4	3	3	3
Errors during development	1	3	4	4	2
Time to implement	2	3	4	4	1
Lines of code to simulate	4(66)	5(6)	4(62)	1(504)	1(614)
Overall ease of use	12	23	24	21	13

Note, that each score is given within a one to five interval. For all parameters five is the best possible score. We will briefly address each parameter which is not intuitively understandable. Startup documentation refers to the 'get started' type of documentation often found within the documentation of development frameworks. Extension options refers to the intuitiveness and availability of methods to extend the framework with custom implementation. Lines of code shows the actual lines it took to implement the model depicted in Fig. 1, denoted within parenthesis after the respective scores (the JaamSim count refers to the script initializing and timing of the JAR (Java Archive) executable). From the gathered parameters an overall ease of use score was given by weighing each parameter equally. Note that JaamSim LoC, while not directly comparable given its different interface, has gotten the highest score since there is no code to write or manage. Another significant aspect of usability in the context is the framework language. Java and python are both very popular languages, and it could be argued that the choice of language would have a greater impact of overall usability. However, since it is very hard to quantify the difference in usability between two high level languages, this has not been directly included in the comparison

5 Discussion

In this section we will briefly discuss some of the most obvious questions in regard to the results of this paper. This includes whether a clear recommendation of a specific framework can be made and a discussion concerning the validity of the testing methodology utilized.

5.1 Comparison

An obvious subject to discuss is whether a concrete recommendation can be made, after evaluating the results. First, it is obvious that if the given load is comparable to the range tested in this project, and the performance parameter is of primary importance, JaamSim could be recommended if the GUI-styled builder is preferable in the given context. Further, SSJ could be recommended if the CLI-styled interface would be preferable, and performance is of greatest importance. However, performance is obviously not the only relevant parameter when choosing a framework. The other parameters tested in this paper related to usability or ease of use can be, depending on the context, at least equally important or even more so than performance. This includes what one could denote as more technical ease of use, such as the extension options and best practice examples (since the user in question in this case is an at least reasonably experienced developer). While it can certainly be argued that the results are of a subjective nature, it will still be of relevance to the aim of this study. The biggest discrepancy between the types of frameworks is in their integration in high level, widely used languages. It could be argued that this is a major advantage of the CLI based frameworks, as the extension and modification opportunities are endless. However, it is not necessarily relevant, depending on the issue at hand. Because of this, no generic conclusion can be drawn, but it is certainly an impactful precursor to an outright recommendation. Lastly, individual developers will obviously have differing preferences in terms of language used, interface, paradigm, etc. In conclusion, no clear recommendation can be made. However, if evaluating from the criteria examined in this article alone, it could be argued that a recommendation could be made for JaamSim if performance is of primary importance and the GUI builder is acceptable, or SSJ if integration directly in the environment of one of the most popular high-level languages would be preferable.

5.2 Test Validity

One almost universal criticism that can be levied in this context of testing a quantifiable parameter such as performance, is that the implementation is subjective, especially since the technologies used are of relatively high complexity. Specifically, the argument could be made that the frameworks performance is directly dependent on the individual implementation. This includes aspects such as integrated parallelization and opportunities for extension (one example of this could be the lower-level language integration in Python for performance-critical modules). This is a valid concern, but we would argue it is not directly applicable in the context of this project. The reason for this lies in the aim of the project, since the evaluation and comparison are geared towards informing on the aspects of these frameworks under what we loosely classify as normal work. While it is hard

to quantify as a term, we can be specific in the methodology used to achieve that aim. First, all implementations follow the 'best practice' implied by their respective documentation, meaning effort was given to implement the tests in the way recommended by the framework's creators. Second, the simulation model was kept quite simple. Lastly, given the holistic nature of the desired results in this project, each test was implemented in a largely similar context to the expected use of the conclusions, meaning a developer with at least basic knowledge of simulation theory and programming methodologies, but with no specific knowledge regarding the given framework.

6 Conclusion

The aim of this study was to identify open-source modelling and simulation libraries and compare them on parameters related to performance and usability. 50 frameworks were initially considered, identified from the semi-structured material collection, which were filtered down on the basis of the following parameters: nature of domain, openness of license, popularity and how recently optimizing or modifying updates were released. After the filtering, five frameworks were extracted for further testing: JaamSim, SimPy, SimJulia, SSJ and JavaSim. JaamSim is a GUI based simulation builder as opposed to SimPy, SSJ, JavaSim and SimJulia which are CLI based, and integrated in Python, Julia and Java respectively.

SSJ was found to perform the best across all intervals in terms of computational time, while utilizing a relatively high percentage of the CPU. Closely following was JaamSim, which also took up a large average percentage of the CPU, especially at smaller input sizes. SimPy was comparable to the aforementioned frameworks for smaller input sizes, but was found to have a higher rate of change in regard to increasing input sizes, scoring significantly lower for the larger input sizes, albeit with a lower percentage of the CPU utilized. JavaSim was found to be significantly slower across all input sizes, but utilized a much lower percentage of the CPU. All aforementioned frameworks were found to have a linear development in the relationship between increasing input sizes and the time needed to process them, but with varying rates. Further, each frameworks' utilization of the CPU and their respective performance on the tested input sizes was found to be corresponding, with the exception of JaamSim in the smaller intervals, likely given the increased overhead of the framework. In the usability comparison SimPy scored the highest, with JaamSim being slightly behind. Lastly, SimJulia scored the lowest in both performance and usability. It was much slower under the given load range and showed worrying internal variation in its performance. Further, SimJulia's documentation was often incomplete and/or outdated, leading to a challenging user experience in implementing the selected model. JavaSim scores only slightly higher in usability, given especially the LoC it took to implement the model.

While much literature pertaining to the subject of this study exists, the landscape of simulation frameworks is so vast and ever evolving that we believe that studies of this type would be relevant to simulation practitioners. However, many of the frameworks that we 'filtered out', as well as frameworks not included in the first place, could be definitely be investigated further, since the parameters we chose to filter on might differ for other researchers. Namely, our focus was on well-supported, open-source and general simulation frameworks.

References

1. Maclay, D.: Simulation gets into the loop. IEEE Rev. **43**(3), 109–112 (1997)
2. Sarkar, N., Halim, S.: Simulation of computer networks: simulators, methodologies and recommendations, presented at the 5th International Conference on Information Technology and Applications, ICITA 2008, January 2008
3. Banks, J., Carson II, J.S., Nelson, B., Nicol, D.: Discrete-Event System Simulation, 5th edition. Upper Saddle River: Pearson (2009)
4. Dagkakis, G., Heavey, C.: A review of open source discrete event simulation software for operations research. J. Simul. **10** (2015). https://doi.org/10.1057/jos.2015.9
5. Franceschini, R., Bisgambiglia, P.-A., Touraille, L., Bisgambiglia, P., Hill, D.: A survey of modelling and simulation software frameworks using discrete event system specification, **43** (2014)
6. Majid, M.A., Aickelin, U., Siebers, P.-O.: Comparing simulation output accuracy of discrete event and agent based models: a quantitative approach. SSRN J. (2009). https://www.ssrn.com/abstract=2830304. Accessed 09 Mar 2021
7. Knyazkov, K.V., Kovalchuk, S.V.: Modeling and simulation framework for development of interactive virtual environments. Procedia Comput. Sci. **29**, 332–342 (2014)
8. Göbel, J., Joschko, P., Koors, A., Page, B.: The discrete event simulation framework DESMO-J: review, comparison to other frameworks and latest development, 100–109 (2013)
9. Mualla, Y., Bai, W., Galland, S., Nicolle, C.: Comparison of Agent-based simulation frameworks for unmanned aerial transportation applications. Procedia Comput. Sci. **130** (2018). https://trid.trb.org/view/1509849. Accessed 09 Mar 2021
10. Barlas, P., Heavey, C.: KE tool: an open source software for automated input data in discrete event simulation projects. In: 2016 Winter Simulation Conference (WSC), pp. 472–483, December 2016
11. Jayatilleke, S., Lai, R.: A systematic review of requirements change management. Inf. Softw. Technol. **93**, 163–185 (2018)
12. SimPy Documentation: 'Overview—SimPy 4.0.2.dev1+g2973dbe documentation' (2021). https://simpy.readthedocs.io/en/latest/. Accessed 09 Mar 2021
13. SimJulia Documentation: Welcome to SimJulia!—SimJulia documentation. https://simjuliajl.readthedocs.io/en/stable/welcome.html. Accessed 09 Mar 2021
14. Jaamsim Documentation: 'JaamSim Free Discrete Event Simulation Software' (2021). https://jaamsim.com/. Accessed 09 Mar 2021
15. L'Ecuyer, P.L., Meliani, L., Vaucher, J.: SSJ: a framework for stochastic simulation in Java, **1**, 234–242 (2003)
16. JavaSim Documentation: 'nmcl/JavaSim', *GitHub* (2021). https://github.com/nmcl/JavaSim. Accessed Mar 2021
17. Sjøberg, D., et al.: A survey of controlled experiments in software engineering. Softw. Eng. IEEE Trans. **31**, 733–753 (2005)
18. Creswell, J.W., Creswell, J.D.: Research Design: Qualitative, Quantitative, and Mixed Methods Approaches. SAGE Publications (2017)
19. Dodig-Crnkovic, G.: Scientific methods in computer science. In: Proceedings of the Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden, Skövde, Suecia. https://www.academia.edu/35111214/Scientific_methods_in_computer_science. Accessed 24 Jun 2021
20. Project Repository: SDU-SimulationFrameworkReviews/SimulationFrameworks. SDU-SimulationFrameworkReviews (2021). https://github.com/SDU-SimulationFrameworkReviews/SimulationFrameworks. Accessed 09 Apr 2021