

Large scale system design aided by modelling and DES simulation: A Petri net approach

Unai Arronategui¹ | José Ángel Bañares¹ | José Manuel Colom¹

Aragón Institute of Engineering Research (I3A), University of Zaragoza, Zaragoza, Spain

Correspondence

Unai Arronategui, José Ángel Bañares, José Manuel Colom, University of Zaragoza, Ada Byron building, María de Luna 1, 50018, Zaragoza, Spain.
Email: unai@unizar.es, banares@unizar.es, and jm@unizar.es

Funding information

Aragonese Government and the European Regional Development Fund “Construyendo Europa desde Aragón”, Grant/Award Number: T35_17D; Spanish Program, Proyectos I+D de Generación de Conocimiento, Grant/Award Number: PG2018-099815-B-100

Abstract

The study of real discrete event systems requires the use of models to cope with complexity and large scale. The only way to understand and analyse their behaviour prior to implementation is, in practice, through distributed simulation. Although it is a widely studied discipline, the difficulty of developing efficient distributed simulation code remains a challenge. The use of model driven engineering approaches allows a smooth way from informal specifications to executable code showing traces of the system behaviour. Formal models allow to conduct the phases of this engineering process, and in this work, the formalism is Petri nets. In the simulation literature, Petri nets have been shown to be particularly suitable for modelling and simulation of discrete event systems. This article reviews the role of Petri nets as the core formalism to support a model-driven engineering approach for the execution of large scale models using distributed simulation. It deals with different aspects related to the Petri net-based languages used at different stages of the modelling and simulation process, from conceptual modelling of complex systems to the generation of code for executing simulations of Petri net-based models. After the review, the article proposes an efficient representation of Petri net-based models. It is analysed from the perspective of the essential properties required for distributed simulation, and was found to provide efficient execution, scalability and dynamic configuration. The article highlights the importance of considering modelling constraints in order to guarantee good properties such as liveness and structural boundedness of Petri net components for the execution of large-scale Petri net models. The Petri net-based methodology is illustrated from the perspective of the impact of the formalism to help developing well-formed models and efficient code for distributed simulation.

KEYWORDS

distributed simulation, large scale models, model driven engineering, Petri nets

Abbreviations: DES, discrete event system; MDE, model driven engineering; M&S, modelling and simulation; PN, Petri net.

This is an open access article under the terms of the [Creative Commons Attribution](https://creativecommons.org/licenses/by/4.0/) License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2024 The Author(s). *Software: Practice and Experience* published by John Wiley & Sons Ltd.

1 | INTRODUCTION

Dynamic systems, whose state is modified in a punctual and distinguishable way in their evolution, are grouped under the term of discrete event systems (DESS). Such systems are of great importance today due to the multitude of relevant applications in our society, for example: health systems,^{1,2} emergency response,³ the Internet of Things,⁴ cyber-physical systems,⁵ or manufacturing systems.^{6,7} As applications become more complex, it is no longer viable to engineer individual self-contained systems but rather to integrate large-scale systems of systems (SoS) involving several domains, for example, smart factories, smart cities, smart power grids, intelligent transportation systems and so forth.⁸ Decision-making and optimisations underlie the choice to build models of several complex scenarios such as those described above.

Dealing with the size of the resulting large-scale SoS model is a major challenge. This scale is the result of capturing several factors and integrating different subsystems. The complexity of today's techno-socio-economic systems demands a comprehensive modelling that encompasses physical and computational interactions, the integration of human behaviour into processes, and the consideration of economic and sustainability requirements. Separating concepts and raising the level of abstraction have proven to be effective software engineering strategies to afford complexity. However, the interrelationship of these separate models is part of the essential complexity of the problem domain. An early separation of the different facets of the system's design makes it difficult to assess the impact of various alternatives.⁹ The construction and simulation of large-scale models that represent the behaviour of complex systems in the most reliable manner is practically the only way to validate them. In this article, for the sake of brevity, the simulation of a model is referred to as the execution of the program that mimics the behaviour of the modelled system. This program can be the result of generating executable code from the model's description, or from an interpreter (simulator) capable of executing the model to generate its behaviour.¹⁰

In addition to the difficulties of handling large size models, there is the fact that they do not fit into a computer for analysis or simulation. Distributed simulation has led to research on large-scale simulations and the acceleration of those consisting of many model partitions running on distributed architectures. However, the translation of the conceptual models into efficient code, for large-scale distributed simulation remains a challenge. There is a large gap between conceptual models that specify the system's behaviour, the structure of components, and the executable code. In software engineering, the model-driven engineering (MDE) is the methodological approach for bridging the semantic gap between the specification of conceptual models and the code generation for an efficient execution on specific platforms. A MDE is supported by a set of languages at different levels of abstraction, tool-based process automation and code generation. Therefore, the natural approach to the modelling and simulation (M&S) of complex systems is MDE for distributed simulation.¹¹

A rigorous model-based approach is required in a formal verification of the design, coding and testing phases in order to detect defects in the requirements compliance. Moreover, it is important to maintain the semantics of model representations from the conceptual modelling to the execution in a distributed simulation. However, formal-model based analysis tools are only useful under certain assumptions. Otherwise, they are insufficient to afford the study of even simple software systems.

In the approach used in this article, a central role is given to Petri net (PN) models, describing the system behaviour and including timing as well as cost information. The goal is to use these models in an intensive way for analysis and simulation purposes. PNs are a family of mathematical models that have been shown to be particularly suitable for modelling (representing) the behaviour of DESSs. In addition to their conceptual simplicity and graphic representation, they can be executed. In PN models the state is represented by variables called places, and the modification of the state is indicated through transitions that change the non-negative integer values stored in the places. The modification of the state is done in a local and atomic way. The automatic analysis of properties is supported by software tools, and when formal analysis becomes impracticable, the model can be executed. Therefore, the synergic combination of simulation and analysis of formal models is necessary for a reliable model design, as well as for an efficient model execution with simulation purposes. The possibility of automating the analysis and execution of PN-based models makes the formalism ideal for supporting an MDE methodology.

Functional and non-functional PN-based model properties can be extracted by means of a qualitative analysis in order to decide whether the model is correct and meets the given qualitative functional properties (e.g., deadlock freedom), and by a quantitative analysis that seeks performance-oriented interpretations of the model such as throughput, utilisation rates, or queue lengths, that will allow to compute reward functions and optimise the system.¹² State explosion can hamper the use of qualitative and quantitative analysis when it is based on the construction of the reachability graph. For complex and scalable DESSs, simulation becomes the only alternative available in practice.

From the modelling point of view, large size PN models require modular and hierarchical PN specifications, which provide more compact and manageable descriptions. However, their interpretation involves costly algorithms.¹³ The execution of large scale PNs models for simulation requires the translation of modular PN specifications into an efficient representation code.

This article is within the context of previous works that specialise in the methodology presented in Reference 12 for the conceptual modelling of DES in distributed simulation following an MDE approach based on PNs.^{1,14,15} The focus is on the PN languages that support the MDE methodology, and the approach in this article is to define the most appropriate PN representations for conceptual modelling and efficient execution in a distributed platform. The proposed code structure for execution preserves the formal semantics of the PN specification translating it into an event dependency network that supports an efficient enabling test and a minimal cost update of the enabled events. Reducing the PN representation to an event dependency network without an explicit state representation provides good properties for distributed simulation such as facilitating model partition and support for separated compilation, scalability, dynamic composition as well as interactions with external agents of legacy simulators.

An MDE approach based on formal models can benefit from a modular analysis of properties. Well-formedness of PN models can be ensured by following some constructive rules. The current paper will show how these constraints allow the definition of basic primitives for the specification of complex models in particular domains and how simple constructive rules facilitate the generation of modular code with good properties for simulation.

In this article, we improve on our previous conference papers,^{14–16} and add some theoretical analysis to generalise the transition enabling characterisation, which simplify code generation and component composition, we focus on the role of Petri nets for modelling and a LEF based coding for large scale distributed simulation, we provide a MDE approach based on Petri nets for the M&S of complex discrete event systems, and finally examples to illustrate the modelling methodology and experimental results using a proof of concept implementation of the simulation engine is presented.

The rest of the article is organised as follows. Section 2 reviews PN-based M&S from a methodological perspective. Section 3 offers a formal definition of *linear enabling functions* (LEF) to characterise the enabling test. The PN encoding based on LEF was introduced in previous works for an efficient interpretation of PN models. The novelty introduced in this article is the utilisation of LEF based coding (LEF-code) for distributed simulation, for which it is particularly well adapted. The translation of a PN component specification into a LEF-coded PN component allows us to bridge the gap between the conceptual model and the executable code. As an additional contribution, this work introduces a new vectorial LEF (VLEF) representation to generalise the transition enabling characterisation. VLEF encoding provides support for separate compilation and the ability to link PN components encoded as LEF once deployed in a distributed execution infrastructure. This feature is essential for modelling and simulating large-scale models. The section highlights additional advantages resulting from the LEF encoding of each transition and the component composition based on transition fusion. These facilitate load balancing among simulators through the exchange of transitions encoded as LEF and enable the derivation of desirable properties, such as boundedness, through component composition.

Section 4 summarises the PN-based MDE approach. Although this work focuses on the languages that support the methodology, particularly on the encoding of PNs to enable efficient interpretation and simulation of large-scale models through the deployment of separately compiled components and their subsequent linking in simulators, this section provides an overview of all the stages in which PN analysis and simulation tools support the entire M&S process. In the context of M&S, an MDE approach involves model transformations throughout the simulation life cycle, including the specification, implementation, and deployment of simulation applications. The peculiarity of the proposed methodology is the use of PNs as the language in all stages, from modelling to simulation. The architectural design is a key aspect that must be taken into account from the beginning for large-scale software systems.

Section 5 introduces examples to illustrate the methodology used for modelling DESs as a set of communicating processes competing for resources. Section 6 shows some experimental results from the proof-of-concept implementation of a LEF-code based distributed simulation engine. Finally, conclusions and lines for future work under this approach are presented in Section 7.

2 | RELATED WORK

The need to build more scalable simulations considering all involved aspects is not a new one. Distributed simulation is a widely used discipline that has promoted standardisation efforts around high level architecture (HLA).¹⁷ The reuse of legacy simulation components is regarded by HLA as the top driver for the adoption of a service oriented architecture,

which is achieved through the mediation of a brokering structure. HLA provides services for information exchange and synchronisation between simulations that together form a *simulation federation*. A detailed explanation of a standardised process for the building of distributed simulation federations can be found in Reference 11.

Tools for code generation of simulation object models, which conform an HLA simulation federation, generally produce skeleton code such as callbacks. These skeletons require manual intervention to implement the behavioural specification of federate components.¹¹ Therefore, large scale models assume the federation of compiled components, and distribution exploits the natural parallelism at the level of model components. The Functional Mock-up Interface* (FMI) defines another standardised interface to be used in computer simulations.

Interoperability and reuse are the main concerns of HLA and FMI. The focus on interoperability leads to neglecting important aspects for efficiency, such as not considering changes in the execution platform, not providing mechanisms to maintain the quality of services on a large scale, such as fault tolerance, or load balancing mechanisms,¹⁸ as well as locking the model partition in components at compilation time. Therefore, it is an orthogonal approach to the methodological aspects considered in this work. As it will be shown later, another important difference of the approach proposed in this article is that the PN model specifications are executable, and therefore a manual intervention is not necessary to obtain an executable component. PN specification is translated into a more adequate representation for efficient execution. This article will focus on composing the behaviours specified by PN-based components, and not on the interoperability aspects considered by an HLA-based simulation federation.

The specification/implementation gap is significant, and there are various contributions and strategies at all intermediate stages of the gap. Below, we review the most relevant works that have been developed, considering that there are no comprehensive proposals like the one presented here, which aims to provide a complete proposal for the gap between specification and implementation in distributed simulation. The main focus of this article is on the PN-based M&S of large size models and the languages supporting the MDE to bridge the gap between conceptual and execution code. To this end, a brief description of PN languages and their characteristics for supporting the methodology is presented.

2.1 | DES formalisms

In Reference 19, a unified modelling framework is presented that encompasses all facets of DESs study. This framework focuses on automata and PNs, coherently linking several topics that have previously been treated separately in the literature. This framework encompasses the modelling, analysis techniques, and control procedures for complex DESs. A comprehensive overview of how these formalisms are used for the control of DESs, covering notions such as controllability, observability, diagnosis, and distributed control, can be found in Reference 20.

Finite state machine (FSM) is a popular formalism for representing sequential systems with a finite number of states. For an FSM-based system specification, it is required the enumeration of all system global states (each one graphically represented with a circle) and the explicit definition of all transitions between pairs of states (each one graphically represented by an arc). Large scale DESs that exhibit concurrency between actions in the system lead to the well known state explosion problem when FSMs are used to specify these systems. Therefore, managing the representation of these large FSMs requires the use of special techniques that consume an enormous amount of resources (if available). FSMs are frequently used in the implementation phase of systems where specific synthesis algorithms are applied to obtain implementations in a particular technology, such as in many domains of hardware systems. Nevertheless, in the design phase of a system where some parameters can be modified in order to adjust the behaviour to the given requirements, starting from an FSM specification can cause a complete rewriting of the FSM to integrate the change (even if it is local). This is a typical situation when a DES must be studied under different scenarios of resource availability. These two difficulties associated with the FSM model, when considering scalable DESs with a high degree of concurrency, make PNs an advantageous alternative.

When advancing in the life cycle of DESs, it is necessary to have methods for analysing the properties of the modelled system for decision-making in the design process that leads to implementation. The use of a model with formal semantics allows the application of analysis techniques supported by computer tools. In the case of FSM, model checking techniques²¹ are widely used but they do not perform well for large systems so they cannot be applied in practice. A systematic literature review of early validation and verification of system behaviour in MDE is out of the scope of the

*<https://fmi-standard.org/>.

article. The interested reader can find it in Reference 22 although it is not focused on simulation of DESs. It is at this point where the availability of executable formal models such as PNs become a necessary tool to proceed with the analysis using DES' simulation techniques, which is advisable to be done on distributed platforms to deal with scalability.

The use of simulation techniques to analyze models is common in many existing formalisms, such as UML Statecharts,²³ PNs, SDL²⁴ and DEVS.¹⁰ These are executable description languages, well-suited for modelling concurrency and supporting good capabilities for hierarchical and/or modular system description. Statecharts have been adopted by object-oriented methodologies and the UML notation, SDL has become very popular within telecommunications, and DEVS becomes popularity for facilitating the integration of discrete and continuous models.

Focusing on the distributed simulation of scalable DESs, PNs is the option adopted in this article due to a crucial characteristic of this formalism: the structure of the net. This structural information makes the state of the model by definition a distributed state (formed by the marking of the places in the net), which facilitates its distribution and maintenance. The structure of the net reveals the causal relationships between the different parts of the model, which enhances its distributability and the ability to exploit information from causally related parts of the model during the simulation. Obtaining useful structural information for the simulation of a PN is possible thanks to the development of a structure theory of PNs, which provides concepts, methods and tools for this purpose.

To address the deficiency in structural analysis compared to automata or PNs, model transformation between formalisms is utilised. This allow modellers the use of tools to automatize the analysis supported for each formalism.²⁵ A process of cross-verification is also supported by the transformation of one formalism to another formalism, and the verification of equivalent simulations.²⁶ The verification of functional correctness of systems using this method is constrained by time-consuming simulative executions of the specification.

Once the good properties of the conceptual model are verified, it can be transformed for simulation to other formalisms,^{23,27,28} simulation tools,²⁵ or languages more suitable for implementation.²⁹ The main difference of our approach is the use of the PN formalism throughout all phases of M&S. In our approach, PN is not only a valuable conceptual model for validation. Structural PN analysis allow an adequate PN coding for efficient distributed simulation, it can be supplemented during simulation time, and refined based on the simulation results. Consequently, it can be associated with the model or its modules in such a way that it can be harnessed in further simulations where these nets will be used.³⁰ Structural analysis of PNs also face practical limitations when dealing with large-dimensional systems. In Reference 30 is presented how to leverage the structure theory of PNs and to address Fujimoto's challenges in parallel and distributed simulation presented in Reference 31 by exploiting the information extracted from the net structure for distributed simulation.

2.2 | PN-based conceptual modelling

A PN has place nodes, transition nodes, and directed arcs connecting places with transitions. Places, transition and arcs do not explicitly describe events, data flows or objects in the system, which makes the formalism inappropriate for conceptual modelling.³² Therefore, it is necessary to specify the PN associated interpretation by defining the meaning of the elements. For example, different semantics are provided to tokens to represent resources, parts, or orders, to places to represent the state of an entity or process, and to arcs to represent data flows or process interactions.

Domain-specific languages (DSLs) facilitate the representation of domain concepts and their relations in a formal manner. For example, in a flexible manufacturing system (FMS) domain the model is composed by physical resources such as robots, buffers, conveyors, tools, and production plans.³³ In the domain of workflow management systems, workflows are made up of cases such a patient in a hospital, an order, or an administrative document that follows a *business process*, and a workflow management system (WMS) provides the *logistics*, that is, the resources to perform the *tasks*.³⁴

The modelling of domain artefacts can be supported by primitives suitable for the resolution of a certain type of model analysis.¹¹ For this purpose, intermediate conceptual modelling languages can be introduced to analyse different perspectives, such as the *resource allocation system* (RAS) perspective in the context of limited resource provisions³³; or the *communicating systems* (CS) perspective, which is a widely known organisational principle for distributed systems.³⁵ These perspectives include the necessary primitives (communicating processes and resources) to develop performance,³⁶ scheduling,³⁷ or economical-oriented models.³⁸ *Performance models* are intended to obtain buffer occupancy, execution times or the degree of resource utilisation; *scheduling models* are intended to obtain the allocation of resources and starting operation times; and *economical models* can be developed to support the analysis of scarce resource consumption *agents* whose behaviour is driven by goals that include competition and collaboration strategies.³⁹

2.3 | Modular and hierarchical PN-based modelling

In order to have good models, it is advisable to consider different levels of abstraction, a process of model refinement and different perspectives. The modeller uses different modelling artefacts at each level of abstraction.

Modular and hierarchical PN specifications are the best option for PN-based conceptual modelling because they provide levels of abstraction, the composition of different perspectives, and more compact and manageable descriptions. In Reference 40, a presentation of the main PN structuring mechanisms is provided. PN compositions are based on *place* or *transition fusion*. These mechanisms provide a horizontal composition of PNs and a vertical composition based on component refinement. An alternative mechanism is the *folding* abstraction used by high-level Petri nets (HLPNs). In a HLPN, tokens carry information that may be represented by data structures, and arcs representing preconditions are labelled by expressions, which identify states defined by the value of tokens. Arcs representing postconditions are labelled by expressions which define state changes by means of the modification of the token values. Folding exploits structural similarities and provides a more concise behaviour representation than ordinary Petri nets. HLPN tools such as *CPN tool*,⁴¹ one of the most popular, use both mechanisms: composition mechanisms based on place or transition fusion, as well as folding based on structured tokens. Structured tokens can also represent nets, which in turn can be dynamically composed by the net containing them. This connection is achieved by merging the transitions of the structured tokens with the transitions associated with the token's container location. This is the case of *reference nets* and its M&S *Renew tool*.⁴²

2.4 | Well-formedness of components and PN compositions

Methodologies used to construct PN models of different domains usually have a modular nature, where a bottom-up approach starts by defining the basic primitives. One of the most important challenges of component composition is enabling the derivation of desired properties of the composed component behaviour (such as boundedness and liveness).

In general, the adopted solution is to impose model structural constraints (PN subclasses). These constraints limit the modelling power, while improving the decision power. Therefore, a structurally constrained model can be supported by analytical tools. Simulation is an attractive alternative to analytical methods because it can be used for unrestricted classes of PNs. However it is important to remember that the focus of this work is on the M&S of large scale PN models. It implies that the simulation will involve intense consumption of computing resources, as well as many executions for the modeller to be confident in the model. The proposed approach is a methodology that deals with the complexity of large-scale models, although, the modelling languages do not impose the methodology. In this sense, place and transition fusion are allowed for component composition, but as it is shown in Section 3, execution code composition is based on transition fusion, which allows us to derive good properties for the execution of model compositions with simulation purposes.

Fortunately, there are different PN subclasses in the literature whose modelling power is adequate for a specific domain. *Well-formedness* of PNs (boundedness and liveness) can be preserved by following some constructing rules. For example, PN-based methodologies for FMS use a RAS perspective of a modular nature to construct PN models, where modules are production plans composed via the fusion of a set of shared resources. Production plans are modelled as strongly connected state machines.³³ Workflow modelling uses the data-flow perspective as the most prominent.³⁴ The process definition consists in the composition of AND-splits, AND-joins, OR-splits, and OR-joins which results in a free-choice PN. The use of a set of transformation rules for free-choice PNs preserves well-formedness.⁴³ Finally, communicating sequential processes is a widely organisational principle for distributed systems. Constraining the modelling to *deterministically synchronised sequential processes*, where process co-operation is organised by storing the producer output in private buffers that respect the consumer's internal choices until they accept the input, guarantees stronger analytical results in the PNs. However, this model is not appropriate for the representation of competition for resources.³⁵ The increase in modelling power, whether to model some competition patterns following an agent perspective,⁴⁴ or to model software applications from a RAS perspective,⁴⁵ reduces the decision power.

2.5 | Code generation for PN interpretation: Enabling test and updating

The translation of a PN specification into executable code is called a PN implementation. Given a PN model of a DES, the *simulation* of the system can be done by *playing the token game*, that is, by moving tokens when transitions are enabled. If a

deterministic or stochastic time interpretation is associated to transitions—timed PNs (TPNs) or stochastic PNs (SPNs)—, the interpretation of the TPN or SPN yields, actually, a discrete event simulation system.

The implementation of a PN can be classified as *compiled* or *interpreted*. The compiled implementation from the PN generates code whose execution traces mimic the sequences in PN. An interpreted PN codifies the structure and marking as data structures used by one or more interpreters to make the PN evolve from a state to another PN state. Compiled implementations have been the option for the development of discrete event control systems.^{46,47}

An interpreted implementation is a model execution based on its interpretation separating the model specification from the simulator, which is essential for scaling simulations and simplifying the dynamic deployment of simulation code on distributed execution platforms.¹⁰ When the model is not wired with the simulator, it enables model portability to other simulators. In simulation contexts where the execution code is not a data structure to be interpreted (e.g., the system is a set of *logical processes* (LPs) compiled before simulation) the execution code is generated at compile-time. LPs define partitions of code that correspond to object or components. Therefore, the level of parallelism and distribution is locked at compilation-time.¹⁶ Even if the best model partition were achieved and deployed, small changes in the hardware configuration or the simulation evolution would produce large imbalances.⁴⁸

The use of micro-kernels specialised in the interpreted implementation of the model avoids having to develop the systems entirely from scratch.⁴⁹ Simulation micro-kernels provide a core invariant portion of services that can be executed in heterogeneous infrastructures such as mini clusters, the cloud, and even allow to embed these simulation services into IoT devices.

The interpretation of a PN model by playing the token game involves a simulation cycle similar to the production system's match-resolve-act cycle with three stages^{13,50}: (1) to test the enabling of transitions; (2) to select enabled transitions to fire (some associated activity can be executed), and decision making in conflicts; and (3) to update the state by moving tokens involved in firings. The first stage is the most time-consuming operation, and the efficiency of a PN interpreter is mainly related to that of the implementation of the transition enabling and transition firing algorithms. In a PN, preconditions are specified by those arcs going from places to transitions, whereas postconditions are specified by arcs going from transitions to places. Transitions whose preconditions are met are enabled and may be fired. When a transition is fired, it removes the enabling tokens from their previous places and puts them in subsequent places.

Two classes of enabling test can be implemented⁵¹: (1) *Place-driven approach*: a place called the representative place of the transition is selected to verify whether the number of tokens in this place is lower than the weight of the arc that connects it to the represented transition. The remaining input places of the transition (synchronisation places) are only tested in order to verify if the transition is finally enabled when the representative place has enough tokens. A representative place is good when its output transition is enabled when marked. Place driven approaches were introduced in Reference 52, and they have been extensively considered under different names.^{47,53,54} (2) *Transition-driven approach*: The search for enabled transitions is based on testing the value of a function (enabling function). Transition driven approaches have only been efficiently applied to binary PNs.⁵³ In Reference 55, a transition-driven method was introduced for weighted place/transition PNs (P/T nets) to characterise the transition enabling by means of a LEF. A comprehensive view of the LEF characterisation of transitions requires a transition classification as well as model transformations. These local transformations add new transitions to the original net, introducing an overhead for the simulation algorithm that reduces, in some cases, the possible advantages of the method. A combination of LEFs with place-driven approaches is proposed in Reference 56 in order to solve these problems for some classes of transitions.

This article introduces a simpler and more suitable LEF characterisation for dynamic composition and distributed simulation. The enabling function is built from the marking of the input places of the transition and the weights of the corresponding arcs. Additionally to the LEF, an event dependency network is built in the compilation process to update the enabling functions of those transitions whose input places marking has been modified.

Finally, HLPNs facilitate the modelling of large-scale models. However, their interpretation introduces costly pattern matching or unification algorithms to evaluate enabled transitions.¹³ Although optimisations of the enabling test of HLPNs have been developed around the unification algorithm,^{13,57–59} for an efficient PN model execution, it is necessary to unfold and flatten the HLPN. This phase is called the *elaboration* process because it is similar to the process of flattening the design hierarchy in VHDL, which results in a flat collection of signal nets and processes suitable for simulation.⁶⁰

2.6 | TPN-based distributed simulation

There is substantial work on distributed simulation of timed place/transition nets (TPNs),^{61–66} which shows how formalism can contribute to the efficient implementation of distributed discrete event simulations. These previous works focused on good partitioning based on the PN structure and synchronisation algorithms. In these works, the TPN is decomposed into a set of LPs assuming a FIFO communication. The LP interface is defined by a subset of places, and the arcs connecting these places are replaced by communication channels. LPs interact exchanging time-stamped messages that represent token transfers. Each LP executes a simulation engine that implements the same simulation strategy to interpret the PN partition and to preserve causality with events simulated by other LPs. The PN structure can also help to obtain lookahead for simulations, and to keep the necessary conflict-resolution information local.^{62,67}

2.7 | Software architecture, deployment and configuration languages: Specifications and implementation refinements

According to the ISO/IEC/IEEE 42010 standard, the software system's architecture is the fundamental document for developing a complex system throughout its entire life cycle. It embodies the design choices concerning the overall structure and behaviour of the system and facilitates the evaluation of system qualities during all decision-making phases, from conception to design and implementation. It describes components, relationships, environment interaction, and guiding design principles. As mentioned earlier, HLA is the architectural proposal for distributed simulation with a primary focus on interoperability and reusability. The design and implementation of reactive systems require the refinement and evolution of the software architecture structure and a clear understanding of the system's intended behaviour.⁶⁸

In the case of the construction of a model for simulation the behaviour's description is the central aspect. However, it is not considered in HLA, which makes difficult to evaluate design choices concerning the behaviour of the system in all life cycle stages. The focus on interoperability and reusability is concerned with component interfaces conformance, and the semantic and format of interchanged data for basic simulation services. The behaviour of components is considered an implementation issue that must be completed by the programmer.

As the software architecture evolves it requires the incorporation of low level details such as the execution platform and model partitions. Pnueli and Harel⁶⁸ consider the evolution and refinement of the software architecture in two different dimensions: (1) refinement of software components and design, and (2) the implementation dimension that requires the adaptation of the behaviour to the specific execution platform.

How to breakup the system's behaviour into components and how to deploy them into the executing platform correspond to the implementation dimension. Pnueli and Harel postpone the problem of connecting the behaviour description of reactive system with their implementation ones and establish the following requisites for any satisfactory method for a behavioural description: (1) it should provide well-structured descriptions with a clear semantic, and (2) it should minimise the dependence on any implementation issues. PNs is a formalism with well defined semantic recommended for use for architectural design.^{69,70} The main advantage is that it is an executable specification that allows execution from the early stages of software system design.

Our hypothesis in this article is that the main challenges of distributed DES simulations pointed in Reference 71 are related with the connection of the behaviour description of simulation models with their implementation, that is: (1) the definition of **modelling languages** allowing the generation of *efficient parallel and distributed simulation code*; (2) the statement of a **clear execution semantic of the model**, and the **execution policy** of its interpreter.^{72,73} They must be oriented to a distributed implementation; (3) the availability of load balancing mechanisms to cope with the unpredictability of the underlying execution environment^{74,75}; and (4) the incorporation of the economic cost and energy consumption, in addition to the traditional speed-up metric in distributed simulations.

Additionally to the need of a executable code that can be easily decomposed, a distributed simulation requires writing the configuration specification in a *configuration language*.⁷⁶ A complete configuration specification includes the components, how these components are connected, and where they are located in the execution platform. To build large-scale distributed simulations it is necessary to decompose the conceptual model into components that can be separately compiled to generate executable code for simulation. The OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) provides a language to describe and manage complex cloud. The management protocols of TOSCA can be modelled by means of Petri nets.⁷⁷

While *structuring mechanisms* in conceptual models are oriented to manage large models, *configuration languages* are designed to deploy executable code in a distributed environment. Conceptual and configuration languages share hierarchical and model composition provided by a declarative knowledge representation scheme. Examples of these structuring declarative specifications are the DEVs *System Entity Structure* (SES),¹⁰ the entities and architectures of VHDL, the *CONIC* programming language,⁷⁶ or the *Page level scripting of TPN Designer* to compose PN models.⁷⁸ These structuring languages define the interface of components including input and output port specifications, primitives for composing component behaviours by means of the connection of entry and exit ports representing event interactions, and a hierarchical composition of components. In the next section, it is presented a PN representation of the modelled system behaviour oriented to an efficient distributed simulation bringing the gap between the executable specification and the implementation dimensions.

3 | EFFICIENT CODE GENERATION FOR PNS DISTRIBUTED SIMULATION

On the one hand, focusing on the properties required for distributed simulation, the current paper points out the advantages of a model execution based on a model interpretation. On the other hand, HLPNs are more appropriate for modelling complex systems, but their model interpretation is inefficient. Therefore, it is assumed that a model flattening process (elaboration) has been done resulting in a P/T net. For this reason, when this article mentions code generation, it refers to the adequate representation of flat P/T nets to interpret the model. From a semantics point of view, the code must preserve the behaviour specification. From an operational point of view, it must provide an efficient model interpretation, facilitate the initial deployment and accommodate an evolutionary change of distributed simulation.

3.1 | LEF-coded flat P/T net models

For an efficient interpretation of PN models, a compiler translates a P/T net specification into an event dependency network based on the idea of LEF.⁵⁵ A LEF value allows to characterise the enabling of a transition (an event that can occur) with a simple linear function of the marking. The LEF characterisation of transition enabling applies to general PNs, even with inhibitors arcs. A LEF of a transition t is a function $f_t : \mathbf{R}(\mathcal{N}, \mathbf{m}_0) \rightarrow \mathbb{Z}$, that maps each marking \mathbf{m} belonging to the set of reachable markings, $\mathbf{R}(\mathcal{N}, \mathbf{m}_0)$, to an integer, in such a way, that t can occur for \mathbf{m} , iff $f_t(\mathbf{m}) \leq 0$.

To define the LEF of transitions, in Reference 55 a transition characterisation in different classes is presented. The classification is based on the number of input places with a structural bound greater than the weight of their respective arcs. The *structural marking bound* (SB) (number of tokens) of a place can be computed as a linear programming problem in polynomial time (on the net structure size) before the compilation time.⁷⁹

A PN \mathcal{N} is said to be structurally bounded if all places are structurally bounded for any initial marking \mathbf{m}_0 . This is a desired and important property to validate the model because places represent limited model resources, such as memory or buffers, and an unbounded place in the model can produce a memory overflow during simulation. Considering large-scale models, the calculation of SBs can be done at component level, and easily extended to large-scale models following some methodological approaches.

Compared with the characterisation presented in Reference 55, this article reduces the classification to two classes and avoids the interpretation overheads introduced by the model transformations presented in the previous work to obtain the LEF function:

- **Class 1.** Transitions with at least one input place, and at most one input place with a SB greater than the weight of its respective arc.
- **Class 2.** Transitions with at least one input place, and at least two input places with a SB greater than the weight of their respective arcs.

It is not possible to characterise the enabling of a Class 2 transition by a single linear function of the net marking. Class 2 transitions enabling can be characterised by means of a family of LEFs. In it, each LEF determines the contribution of a subset of input places to the enabling of the transition. The minimal number of LEFs is equal to the number of input places with an SB greater than the weight of the arc connecting it to t . A *Minimal Legal Covering* is configured with a

subset of places for each place with an SB greater than the weight of the arc. The rest of the input places are distributed discretely among these subsets. Therefore, different configurations are possible. Every subset can be characterised by a LEF. The set of functions can have a vectorial representation called **Vectorial LEF** (VLEF). The firing of a transition t of Class 2 can occur at a marking \mathbf{m} iff the value of its VLEF for \mathbf{m} is less than or equal to the vector zero, $f_i(\mathbf{m}) \leq \mathbf{0}$ (i.e., $\forall j \in \{1, \dots, k\}, f_i(\mathbf{m})[j] \leq 0$).

In the following examples, for simplicity and clarity in the explanations, all transitions have their input places with structural bounds lower than the weight of their respective arcs. In this case, LEFs adopt the simplified form $f_i(\mathbf{m}) = \sum_{p \in \bullet t} \text{Pre}[p, t] - \mathbf{m}[p]$, equivalent to the down-counter of a transition in the simulation method called enabled transition method.⁵⁶

Figure 1 shows P/T nets in components that will be used in the next subsection to illustrate component composition of PNs based on the fusion of LEF-coded transitions. For now, for the sake of clarity, the components and composition details will be ignored and the work will focus on PNs as if they were independently defined. For example, for transition t_1 in component `sub1` in the Figure 1, the LEF is: $f_{t_1}(\mathbf{m}) = 1 - (\mathbf{m}[P1])$, $\forall \mathbf{m} \in \mathbf{R}(\mathcal{N}_{\text{sub1}}, \mathbf{m}_0)$, where \mathbf{m}_0 is the initial marking depicted. At \mathbf{m}_0 , place $P1$ doesn't have any tokens, the value of $f_{t_1}(\mathbf{m}_0) = 1 \not\leq 0$, and the t_1 transition is not enabled at \mathbf{m}_0 . However, the initial LEF value of t_0 in component `sub0` is defined by the linear function $f_{t_0}(\mathbf{m}) = 1 - (\mathbf{m}[P0])$, $\forall \mathbf{m} \in \mathbf{R}(\mathcal{N}_{\text{sub0}}, \mathbf{m}_0)$, its value for the initial marking is $f_{t_0}(\mathbf{m}_0) = 0$, and it is therefore enabled.

The use of LEFs, as previously presented, for the characterisation of the enabling of a transition requires an explicit representation of the marking of the net and the LEF itself as a marking function for each transition. In the case of a distributed simulation, this causes two problems that make this execution model of a PN not well-adapted for this purpose: (1) the explicit representation of the marking in a distributed environment is a set of variables shared by a set of distributed simulation engines. It requires mechanisms to maintain the coherence and consistency of the marking variables (which is in fact a bottleneck for distributed simulation); (2) the functional representation of the LEF requires a continuous evaluation for the marking in order to determine the enabling of a transition.

To address these two problems, the LEF mechanism for a transition is implemented according to the following principles: (1) the current *value* of the LEF is stored exclusively in the simulation engine that owns the transition in the simulation (initially this value corresponds to the value of the LEF at \mathbf{m}_0 and computed in compilation-time); (2) each time a transition occurs in the net, a constant is sent to each transition whose enabling is being affected by its occurrence. This constant is used for the updating of the LEF value of the affected transition.

In this way, the LEF data structures represent an event dependency network, where the marking of the net and places has been removed from the PN representation. It is important to note that this feature of LEF encoding makes it particularly suitable for the simulation of distributed systems, as it eliminates access to shared memory. With this strategy, the explicit representation of the marking and the re-evaluation of the LEF are not needed. The changes of a LEF are based in the constants sent by the transitions that modify its enabling conditions. That is, if $\mathbf{m} \xrightarrow{t'} \mathbf{m}'$, $f_i(\mathbf{m}')$ can be computed from the value of $f_i(\mathbf{m})$, and from a static parameter known at compilation-time that represents the change of f_i after the occurrence of t' . This parameter corresponds to changes in the contents of tokens of the t input places as a consequence

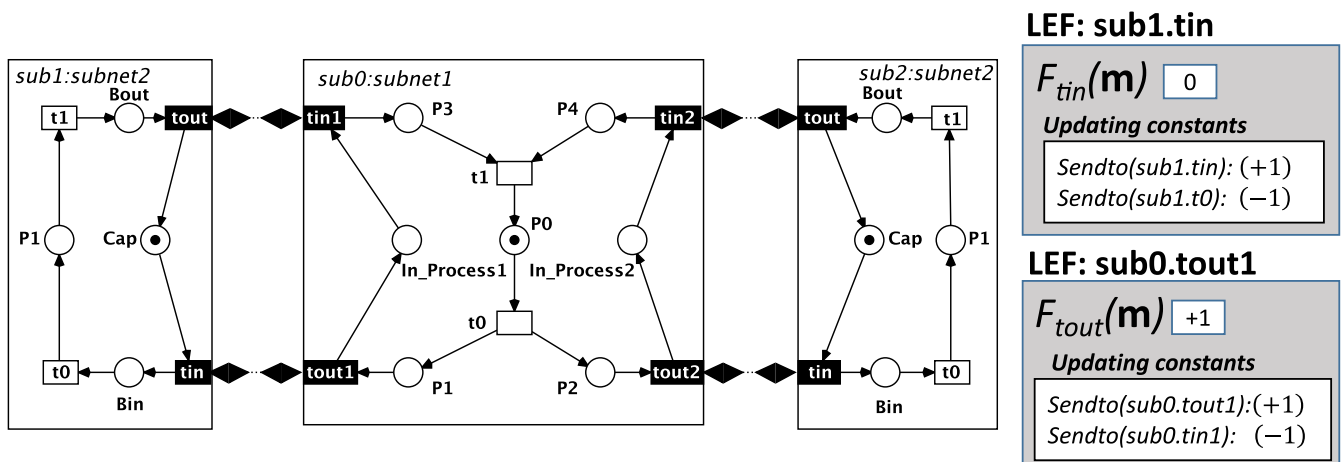


FIGURE 1 Petri net components and LEF codification of transitions.

of the occurrence of t' . Thus, the updating equation for any LEF when t' occurs is $f_t(\mathbf{m}') = f_t(\mathbf{m}) + UF(t' \rightarrow t)$, where $UF(t' \rightarrow t)$ is known as the *Updating Factor (UF)* of t' over t obtained from the structure of the net and its initial marking. According to the previous comments, the compilation of a P/T net produces a representation of the net where there is an entry for each transition t grouping: (1) the variable maintaining the current value of the LEF and initialised to $f_t(\mathbf{m}_0)$; and (2) the list of updating factors (simulation's events), $UF(t \rightarrow t')$, that will be sent to each $t' \in (\bullet t) \bullet \cup (t \bullet) \bullet$ whose enabling conditions have been affected by the occurrence of t .

The enabling characterisation of class 2 transitions, which are defined by a set of functions, is represented by a vector of LEF structures. Taking into account these new VLEF structures, in all transitions the UFs addressed to VLEF-coded transitions are represented by vectors of UFs accordingly.

The partition of a model for a distributed simulation only requires defining the set of transitions to be grouped in each of the distributed simulation engines and loading the previous data associated to transitions in the corresponding engine. The information associated to each LEF/VLEF encoded transition is independent of the information of any other transition.

3.1.1 | LEF-coded P/T net component composition

As previously presented, conceptual hierarchical modelling can use place or transition fusion for component composition. To support the scalability of distributed simulation, large-scale models require separate compilation and further linking of the components that are distributed in different nodes. In addition to the characterisation of class 2 transitions, the idea of VLEFs can be used to facilitate distributed compiled component linking. The VLEF data structure, which represents a set of LEFs, can be useful to represent a transition fusion easily obtained from the collection of initial LEF/VLEF data structures of transitions.

The composition of structurally bounded PNs by transition fusion has the advantage of resulting in structurally bounded PNs.⁸⁰ It is a good property to ensure that LEF values will be bounded during the simulation. For this reason, in this methodological approach, transition fusion is the option chosen for component composition and linking compiled code, as well as for the conceptual modelling in order to ensure boundedness.

Left Figure 1 shows two components, `sub1` and `sub2`, which are instances of `subnet2`, and the middle component `sub0` which is an instance of `subnet1`. Interfaces for composition by transition fusion are graphically represented by black transitions with a diamond symbol. Fusion operations are represented by the dotted lines that connect interface transitions. The graphical representation of entities is similar to DEVS¹⁰ or TPN designer⁷⁶ tools' notation with boxes representing entities, and input–output arrows (black transitions in the interface) representing ports. In contrast to these previous tools, the diamond symbol represents a *rendezvous*, and therefore there are not oriented port mappings. We will denote in the text transition fusion with the symbol $\langle \rangle$. The bottom left figure shows the resulting flat PN.

Each one of these components can be compiled separately. In the example, right Figure 1 shows the LEF code of transitions `sub1.tin` and `sub0.tout1`, which result respectively from the separated compilation of the `sub1` and `sub0` components. All transitions of components are of Class 1, and can be represented by LEF code, which is illustrated by the labelled boxes that contain the current LEF value, and the list of UFs (scalars) to be sent to the transitions affected by the occurrence of the owner transition.

The components composition results in transition fusions `sub1.tout` \langle `sub0.tin1`, `sub2.tout` \langle `sub0.tin2`, `sub0.tout1` \langle `sub1.tin` and `sub0.tout2` \langle `sub2.tin`. The naming convention for hierarchical entities follows the usual dotted notation. URI conventions can be used to assign a unique identifier to distributed components upon deployment. Bottom right figure shows the resulting `CompositionNet` flat PN. For example, the `sub0.tout1` \langle `sub1.tin` mapping specifies that transition `sub1.tin` will be fused with transition `sub0.tout1`. Uri.

If component composition is specified before it is compiled and the size of the model allows the compilation in a machine, the model can be flattened and then compiled. The flat model that results from the component composition of Figure 1 is shown in Figure 2. The resulting LEF code of fusion transition `sub0.tout` \langle `sub1.tin` is shown in the top right position of Figure 2. In this example, all transitions are of class 1, and therefore if the model is flattened and compiled in one machine, all LEF values are scalar.

Alternatively, if the components are separately compiled, the steps to configure the data structure of a VLEF resulting from the fusion of two LEFs (or VLEFs) compiled transitions are: (1) LEF values are stacked up in a vector of LEFs. The order in the mapping declaration is used to interpret the index of the resulting VLEF value of transition fusion. (2) For

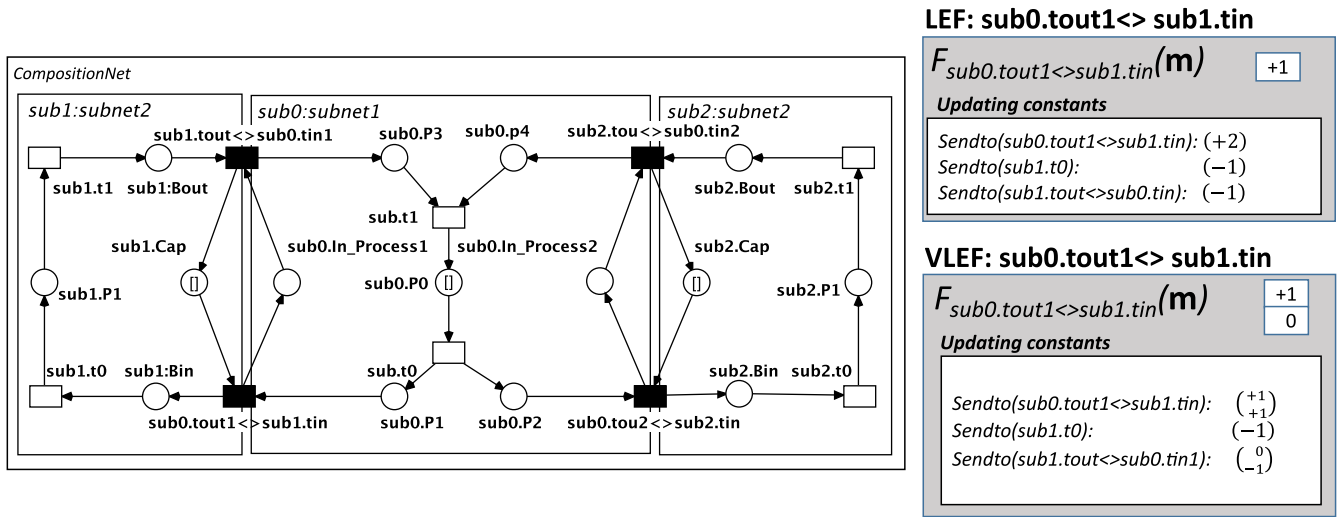


FIGURE 2 Flat model of a component composition with alternative codification of transitions. VLEF resulting from linking separated compiled components, and alternative LEF code representation after model flattening.

each mapped t , and for all $t' \in (\bullet t) \cup (t \bullet)$, search the UF for t in the list of t' , and substitute it by a 0 vector, where the index specified by order of t in the mapping, is replaced by the $UF(t \rightarrow t)$. (3) Retain the list of UF s of fusion transitions. If there are two UF s with the same destination transition, substitute them by a unique UF resulting from the addition of the two ones found.

In the bottom right position of Figure 2 it is shown the alternative VLEF data structure resulting from the transition fusion specified by the mapping $sub0.tout1<>sub1.tin$. The first step of fusion results in the vector value $(+1, 0)$. The mapping specifies the first index of the VLEF value from $sub0.tout1$, and the second one from $sub1.tin$. In the second step, the UF of $sub0.tout1$ $Sendto(sub0.tout1): (+1)$ is replaced by the vector $(+1, 0)$, and the UF of $sub1.tin$ $Sendto(sub1.tin): (+1)$ by the vector $(0, +1)$. As $sub0.tout1$ and $sub1.tin$ share the same identity after the transition fusion, the resulting UF $Sendto(sub0.tout1<>sub1.tin): (+1, +1)$ results from the addition of the two previously obtained vector UF s. Step 3 completes the list of UF s with the retaining of the rest of UF s ($Sendto(sub1.t0)$ and $Sendto(sub0.tin1)$). Finally, $Sendto(sub0.tin1)$ is replaced by $Sendto(sub1.tout<>sub0.tin): (0, -1)$, which is the result of step 2 for the fusion of transitions $sub1.tout<>sub0.tin1$.

The resultant LEF/VLEF code representation of a transition fusion must be deployed in only one of the components when they are distributed.

VLEF encoding is well suited for the simulation of large-scale PNs. It offers support for modularity in describing the model, and furthermore, facilitates the compilation of these models to generate the code that will ultimately be loaded into various simulation engines. The concept of modular and hierarchical component-oriented PN descriptions that allow models to be built and described in manageable module dimensions is not new, as demonstrated in the related work. However, compiling one of these descriptions, which can contain millions or even tens of millions of transitions, presents a significant challenge that requires substantial computational resources (both in terms of time and memory). As a result, the practical limitations on the sizes of PNs that can be simulated are severe, undermining one of the objectives of distributed simulation even before the simulation can be initiated.

To address this issue in model compilation, VLEF encoding provides suitable mechanisms to support the separated compilation of modules or parts of the model, each of manageable size. These modules can be linked together when loading the model into the simulation engines.

3.1.2 | LEF-coded TPN distributed simulation

Previous subsection has presented a LEF-coded P/T net representation for efficient test-enabling and updating enabling state. In P/T nets, these actions are atomic, instantaneous and indistinguishable. TPNs introduce firing delays to

transitions. The model interpretation of TPNs requires dealing with the problem of *race conditions*, providing support for temporal interpretation of the model (*firing time, enabling time*), and interpreting the concurrent firing of transitions (*single server, or multiple server semantics*).⁷² For simplicity, this article considers single server semantics. Race conditions and the temporal interpretation of TPN makes it necessary to perform the transition firing in two phases: immediate and projected update. It is reflected in the LEF data structure where UFs are decomposed into immediate and projected lists.

To explain a distributed simulation based in the LEF-coded TPN interpretation, Figure 3 reproduces the example presented by Chiola and Ferscha in Reference 81. T_1 represents the occurrence of a *machine failure* event, and T_2 the repair rate. *Future* t_1 and t_2 values represent respectively the sequence of exponentially distributed random times ($\exp(\lambda = 0.5)$) for T_1 and T_2 .

To exploit concurrency, the model example is partitioned in two PN components (dashed boxes). Following the traditional nomenclature in distributed simulation, these partitions together with their interpretation engines are called *logical processes* LP_1 and LP_2 . To highlight the fact that simulation engines can dynamically interchange LEF-coded transitions, and therefore the partition is not locked in a compiled LP , the current work designates *Simulation process (SP)*, or *Simbot* as the pair made up of a model partition and a simulation engine. In this example, it is assumed that the complete PN has been compiled generating the LEF-coded transitions illustrated in the figure, and that each transition has been deployed in different *Simbots*. The only constraint that has been considered to distribute transitions is that conflicting transitions must reside in the same *Simbot*.⁶²

Figure 3 shows the extended LEF codification associated to each transition t' for the simulation of TPNs: (1) **Identifier** of t' , which is a *global name* recognised in all sites of the simulation process; and the **coupled conflict set (CCS)**. A CCS is a structural relation that groups transitions which share some previous input place. The coupled conflict relation is defined as the transitive closure of the structural conflict relation. The equivalence class (or coupled conflict set) of transition t' is denoted by $CCS(t')$ ⁷⁹; (2) $\tau(t')$. **Firing time** associated to transition t' . It stands for the duration time of the action associated to the occurrence of t' ; (3) **Counter**. Variable containing the current value of the LEF (or values of VLEF) initialised with $f_{t'}(\mathbf{m}_0)$; and updated whenever the transition—or a transition affecting it—occurs, according to the received updating factor; (4) **Immediate Updating List (IUL)**. Set of transitions $(\bullet t')$ whose LEFs/VLEFs must be immediately updated after the occurrence of t' containing the corresponding UFs to be sent [$(\bullet t')$ includes t'], and (5) **Projected Updating List (PUL)**. Set of transitions $(t' \bullet)$ whose LEFs/VLEFs must be updated after the firing time of t' , which contains the corresponding UFs to be sent.

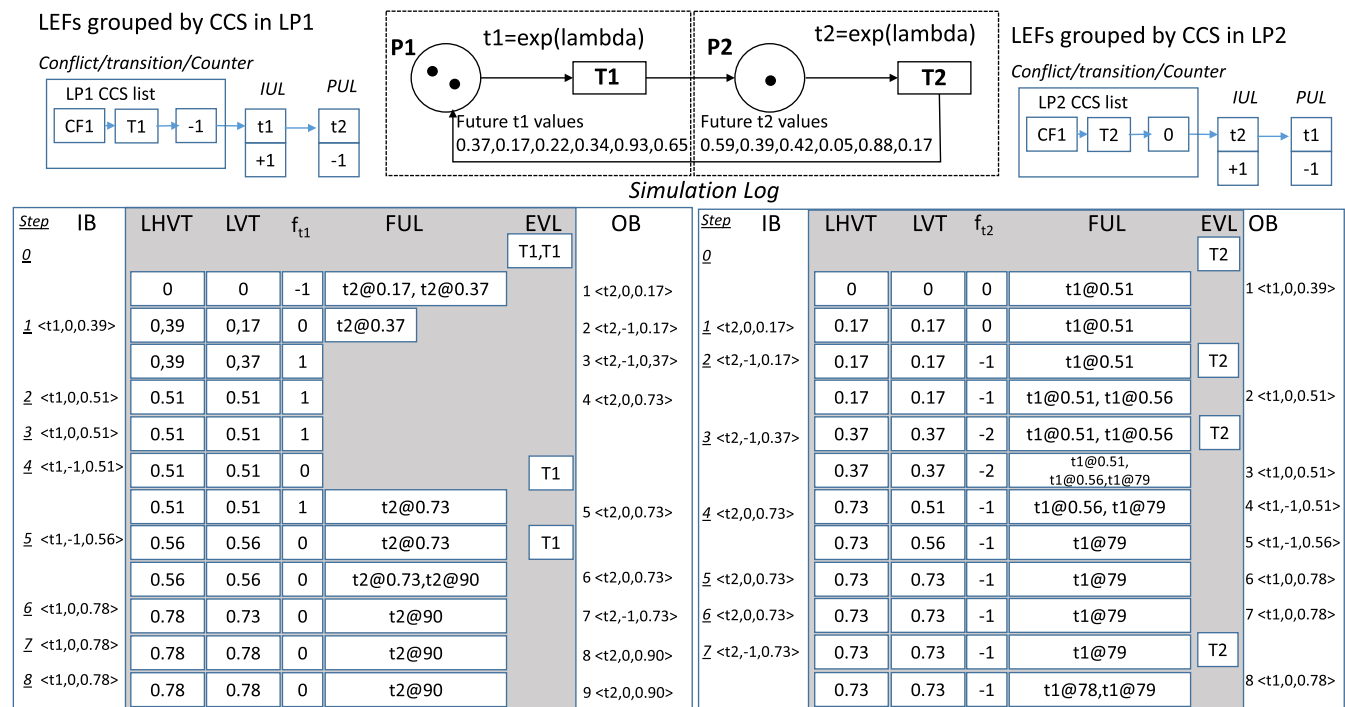


FIGURE 3 Distributed simulation log reproducing the Chiola and Ferscha sample presented in Reference 81, and illustrating event interaction (interchanged UFs) between two Lef-coded PN partitions.

Distributed simulation introduces the need for protocols that prevent causality errors. A multitude of such protocols for time synchronisation have been described in Reference 82. The execution model presented here and the mechanisms for its distributed implementation are the core of the development of any distributed simulation strategy, either conservative or optimistic.⁸³ The distributed simulation is coordinated by means of time-stamped messages between PN partitions, which represent how to update the enabling of external transitions when a transition fires.

Algorithm 1 shows a conservative distributed *simulation manager* sketch that invokes the interpreter algorithm, and Algorithm 2 implements the *simulation interpreter*, which is executed until the *local virtual time (LVT)* reaches the *local horizon virtual time (LHVT)*.¹⁶ The *LHVT* or *lower bound time stamp* is the minimum time stamp message received from all adjacent *Simbots*.

Algorithm 1. Conservative distributed simulation manager.

```

1: when Start() is received                                     ▷ Initialise Simbot
2:  $VT \leftarrow 0$ ;  $FUL \leftarrow \{\}$ ;
3: for all ( $t \in PUL_{ext}$ ) do                                       ▷ Send lookahead to all adjacent Simbots
4:    $Adj[t] \leftarrow 0$ ;
5:    $t \ ! \ \langle 0, lookahead(t) \rangle$ 
6: end for
7: for all ( $t \in LEFs$ ) do                                           ▷ Initialize the Event List (Enabled transitions list)
8:   if ( $f_i(M) \leq 0$ ) then insert( $EVL, t$ );
9:   end if
10: end for
11:
12: when Event( $t, UF, ts$ ) is received                                   ▷ Event Received
13:  $Adj[t] \leftarrow ts$ ;
14: insert- $FUL(t, UF, ts)$ ;
15: if allReceived( $Adj$ ) then                                       ▷ When time stamp messages are received from all adjacent Simbots,
16:    $LHVT = \min(Adj)$ ;                                               ▷ it simulates until the LHVT
17:   Simulate( $LHVT$ );
18:   for all ( $t \in PUL_{ext}$ ) do                                       ▷ Once reached LHVT, it sends messages to adjacent Simbots
19:     if ( $t \in FUL$ ) then
20:        $t \ ! \ \langle UF, ts \rangle$ ; remove- $FUL(t)$ ;
21:     else
22:        $t \ ! \ \langle 0, lookahead(t) \rangle$ ;
23:     end if
24:   end for
25: end if

```

The bottom Figure 3 shows a sample log (events recording) of the conservative distributed simulation. Grey background shows the variables of *Simbots* executing LP_1 , and LP_2 , and white background shows the input and output buffers (IB , OB), the time-stamped messages with the interchanged UFs, and the steps executed by the respective distributed simulation managers.

For simplicity, the **simulation manager** in Algorithm 1 shows only the reactive behaviour when an event is received in an actor-like style. The **Start** message (lines 1-10) initialises the *simulation manager*. It initialises the Event List (EVL) that contains the initial list of enabled internal transitions (lines 7-10), and also initialises to zero all time stamps received from adjacent *Simbots* (Adj). It sends the lookahead value to each transition in PUL_{ext} as well, which contains the list of transitions in adjacent *Simbots* that can be affected by the firing of transitions in the *Simbot*.

When an **Event** message is received, the *simulation manager* executes a simulation step of the *simulation interpreter* Algorithm 2: the first step of the *simulation manager* is to update the received time stamp of transition in Adj (line 13), and it translates the external event of the *Input Buffer (IB)* into the *Future Updating List (FUL)* (line 14), which plays the role of the future event list in an event-driven simulation. The function *insert-FUL()* maintains events ordered by time stamp. Every event in the FUL has a pointer to the transition t to be updated, the updating factor $UF(t' \rightarrow t)$ delivered by each fired transition t' ($t \in (t' \bullet)^*$), and the *time* at which the updating must take place. In the log shown in the figure, UFs are removed from the FUL to avoid redundant information. After that, the event processing checks that a message has been received from each adjacent *Simbot* (allreceived(Adj), line 15). In this case, the minimum time stamp (ts , line 16) is used as $LHVT$ to execute a **simulation step** of the interpreter. After this, messages are inserted in the *Output Buffer (OB)* for each transition in PUL_{ext} . Then, the *simulation manager* empties the OB and sends asynchronous messages to all adjacent *Simbots*. These messages allow them to advance their simulations expecting not to receive messages with smaller timestamp in the future. In order to know when transitions in adjacent *Simbots* must be updated, the message contains the t identifier, the UF and the ts (line 20). In case there are not events for the adjacent

Algorithm 2. Simulation interpreter of a LEFs-coded TPN.

```

1: procedure Simulate (LHVT)
2: while ( $LVT \leq LHVT$ ) do
3:   if ( $head - FUL.time > LVT$ ) then  $LVT \leftarrow head - FUL.time$  ▷ Update Virtual Time
4:   end if
5:   while ( $head - FUL.time = LVT$ ) do ▷ Update Event List
6:      $t \leftarrow head - FUL.pt; f_i(M) := f_i(M) + head - FUL.UF;$ 
7:     if ( $f_i(M) \leq 0$ ) then  $insert(EL, t);$ 
8:     end if
9:      $head - FUL \leftarrow pop(FUL);$ 
10:  end while
11:   $EVL \leftarrow Sort(EVL, CCS, Strategy);$  ▷ prioritises transitions in conflict in EVL
12:  for all ( $t' \in EL$ ) do ▷ Fires enabled transitions
13:    if ( $f_{i'}(M) \leq 0$ ) then ▷ Checks transition is enabled yet
14:      for all ( $t \in IUL(t')$ ) do
15:         $f_i(M) \leftarrow f_i(M) + UF(t' \rightarrow t);$ 
16:        if ( $t = t'$  and  $f_i(M) \leq 0$ ) then ▷ Avoids race conditions
17:           $insert - FUL(t, 0, \tau(t) + LVT);$ 
18:        end if
19:      end for
20:      for all ( $t \in PUL(t')$ ) do
21:         $insert - FUL(t, UF(t' \rightarrow t), \tau(t') + LVT);$ 
22:      end for
23:    end if
24:  end for
25: end while
26: end procedure

```

Simbot, the message contains a zero (null) UF value and the lookahead time stamp (line 22). The algorithm can be improved by reducing the number of messages.⁸²

When the *simulation manager* executes a conservative strategy, the model requires to exploit the **lookahead** information to speed up the simulation. The lookahead comes directly from the net structure.⁸¹ To calculate the lookahead for every external transition, the sample uses the precomputed *future list* of firing times. The lookahead of a transition is calculated as the minimum time-stamp of events that reference this transition in the FUL and the times that result from the addition of the times in the future list and the LVT taking into account a number of times equal to the enabling degree of transition.

The **simulation interpreter** in Algorithm 2 executes the interpreter until the $LHVT$. First, it advances the LVT until it reaches the time stamp of the first event in the FUL (line 3-4). The algorithm then updates all LEF values with the UF of events that are in the FUL and have the current LVT . Afterwards, it inserts enabled transitions in the EVL (lines 5-10). In line 9, $pop(FUL)$ pops and returns the head of FUL ($head - FUL$), granting access to the fields of events in FUL using the dot notation. Subsequently, the algorithm deals with all enabled transitions in CCS (line 11), solving conflicts by sorting enabled transitions according to a defined strategy (policy).

Then, the interpreter takes all enabled transitions in the EVL firing enabled transitions in order (lines 12-24). For every enabled transition, the algorithm immediately applies IUF update factors, that is, tokens are removed from input places as soon as a transition occurs (lines 14-19). The algorithm then inserts events into the FUL for subsequent processing using the PUL update factors, which means that tokens will appear in output places in future clock time (lines 20-22). Then, all transitions' LEFs values in the FUL with the same time-stamp of the current LVT are checked, and enabled transitions are inserted in the EVL . As it was highlighted in Reference 16, LEFs make the representation and updating of the marking of the PN model unnecessary. The PN marking can be easily defined collecting a log that contains the occurrence of transitions, where each one of them is labelled with the simulation time. The occurrence sequence and the net state equation (an algebraic computation) can be used to compute the reached marking from the initial one.

Only a mailbox is associated with a *Simbot*, which allows to adapt the algorithm to support dynamic topologies of *Simbots*, or to use optimistic strategies incorporating an exception handling mechanism.

3.1.3 | VLEF-coded PN supported interoperability and simulation federation

VLEF encoding is also well suited for models whose simulation needs to interact with external agents or legacy simulators in order to federate a set of simulators. In some cases, when modelling a given system using PNs, certain parts of the

system cannot be represented solely in terms of nets. For instance, there may be predicates associated with the occurrence of a transition that require consulting external variables to determine their truth value. These external variables can be inputs to the system from an unspecified environment or simply the inputs coming from human agents that must be integrated in the system's behaviour (e.g., decision making processes external to the system). A typical example such a predicate is a function used to resolve conflicts between transitions.

To support the simulation of models in this class of systems, simulation engines need to be equipped with the capability to evaluate predicates associated with transitions, where the truth value of these predicates determines the enabling condition of the transition, but is not based on Petri net terms. The VLEFs allow a natural integration of these predicate-based enabling conditions. To achieve this, the following mechanisms should be introduced in the simulation engines and the data structures associated with the transitions:

- Extend the vectors that represent the VLEF encoding to include an additional entry for each predicate associated with the occurrence of a transition. Instead of storing an integer value, this entry should store a call to an external function. This function evaluates the associated predicate when all the variables involved in the predicate have been obtained. The function will return 0 if the predicate is true (enabling the transition) or 1 if the predicate is false (disabling the transition). By extending the VLEF with these predicate entries, the transition will be enabled only if all its components are less than or equal to zero. Before performing this test, the evaluation functions of the predicates need to be invoked to obtain the vector of values for comparison.
- In cases where constants need to be transmitted to transitions with predicates from transitions that modify their enabling degree, the components related to the predicates are generally set to zero. However, in certain cases, the transmission of parameters to a simulation data repository or a data collection mechanism for decision-making purposes within the simulation can be achieved by including invocation of transmission procedures within these entries. This allows for the seamless sending of data during the simulation process.

4 | THE OVERALL PN-BASED MDE APPROACH FOR LARGE SCALE DISTRIBUTED SIMULATION

The objective of large-scale DES M&S is to obtain the most precise model by coupling all perspective models, and to allow the modeller to observe the effect of decisions in a clearer way. Figure 4 summarises the stages of the MDE approach presented in References 1,14,16:

Conceptual PN-modelling: Modelling with a DSL language that provides the basic conceptual entities of an application in a specific domain. Interactions with the environment or external simulators are also modelled. The conceptual model is the initial product that can be verified before executing the simulation model. A PN-based model can leverage structural analysis to support the **functional verification** process pointed in Figure 4.

As shown in Reference 30, PN models that are well-formed for simulation are those where the resulting net system remains bounded for any initial marking. This requirement, known as the structural boundedness property, must be guaranteed by the net structure. This property can be verified using the incidence matrix C by checking if the following inequality has at least one solution: $\exists y > 0$ such that $y \cdot C \leq 0$ (place-based characterisation).

While algebra software packages can generally handle matrices with tens of thousands of rows and columns on modern systems with sufficient memory, this is relatively limited for scalable Petri net modelling of complex system-of-systems using a modular approach, which may consist of millions of places and transitions. The solution proposed in References 1,16,30 involves obtaining structurally bounded models by construction, starting from small-sized components that each possess this property. The construction process involves integrating these components through operations that maintain the property (closed operators with respect to the class of structurally bounded Petri nets). For instance, components can be Petri subnets that are integrated by merging specific interface transitions defined within each component.

Composition of live components can result in a deadlock. The executable nature of PNs allows the incremental verification of coupled components by simulation. Alternatively, once again, well-formed models can be constructed using a methodology that incorporates constrained PN models tailored to a specific domain. Significant efforts have been to identify subclasses of Petri nets that are suitable for specific domains and facilitate the verification of properties. The well-formedness of Petri nets (including boundedness and liveness) can be maintained by adhering to specific construction rules in these domains that can be found in the literature referenced in Section 2.4.

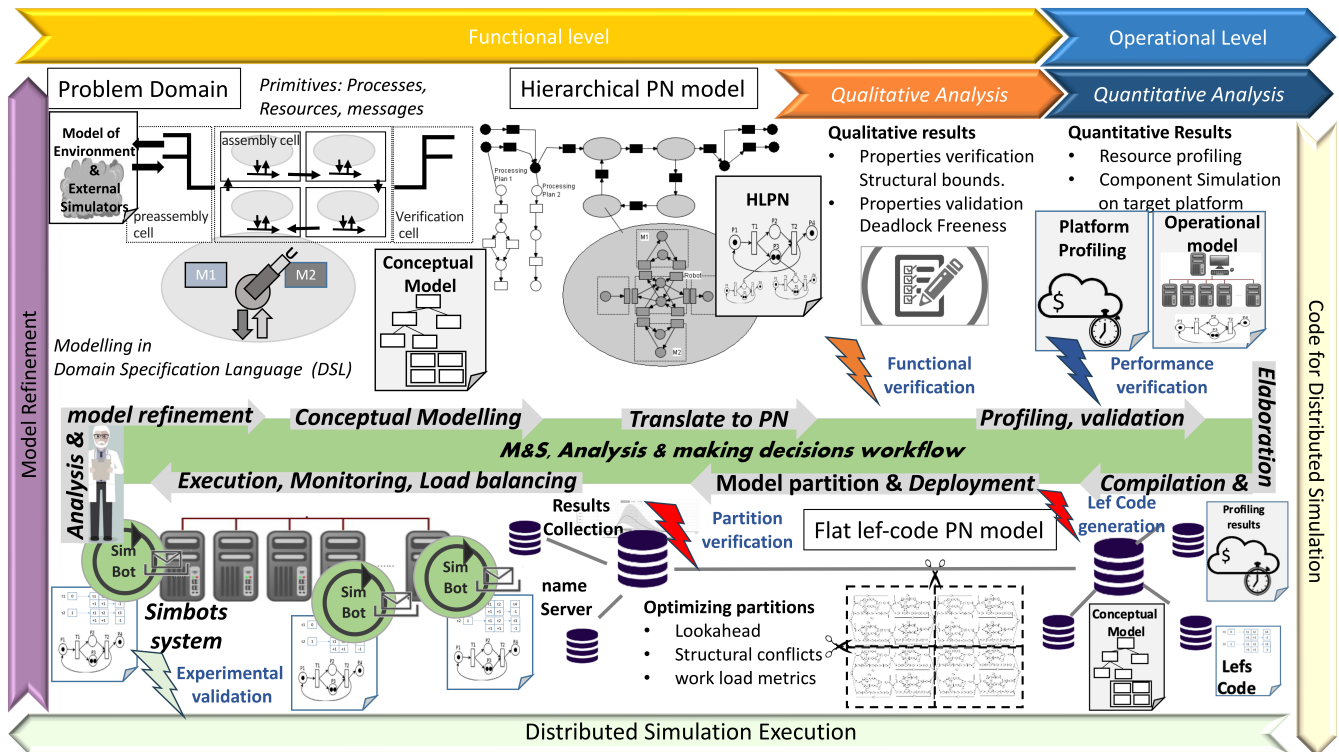


FIGURE 4 Large scale M&S supporting decision-making workflow on a PN-based MDE approach.

Timing and decision model annotations: The model can be annotated with labels that constraint its interpretation. These annotations are required when the model *refinement process* goes down the level of abstraction until it is not possible to achieve a more detailed description.

If the limit in the abstraction level is due to the lack of knowledge, PN models can be annotated and configured with deterministic time, probability distributions, or declarative knowledge that mimics the system's behaviour. For example, when the resource behaviour in a model cannot be detailed, time annotations can be obtained from data acquired by profiling the execution on the resource.⁸⁴ If there is not any information of how decisions are made in conflict resolutions, a rule-based engine can make decisions when the model is not deterministic.

An alternative methodological approach when the execution of large-scale models is not possible is *Recursive simulation*.^{39,85} It consists of conducting simulations at lower levels, which are launched to compute parameters that are useful for higher base simulation.

Operational model: In order to provide assistance in the next phase of deployment and configuration, it is essential to develop an operational model of the simulation platform. The *operational model* enriches the functional model with characteristics of the execution platform to develop a *quantitative analysis*. The efficiency of distributed simulation depends on the implementation of efficient simulation engines, the definition of data structures representing the workload, the number of events interchanged between model partitions, and the dynamic deployment of these workloads among the computational resources. The definition of metrics for the evaluation and estimation of workload is an essential aspect for distributed simulation, especially in the cloud, where computation and communication come at a cost.¹⁵ It makes it necessary to profile components by micro-simulations on resources. A PN-based model labelled with time obtained from the result profiling is essential to perform the **performance verification** pointed in Figure 4. An example of this methodological approach for performance verification can be found in Reference 12. It should be noted that the conceptual model that results from the composition of the functional and operational levels in different domains (FMS, WMS), is considered the functional level for the simulation process.

Elaboration and compilation: An *elaboration* process translates high level PN specifications into a *flat model* that is more adequate for efficient interpretation. The conceptual model is *compiled* into efficient code based on the idea of the LEF summarised in Section 3. **LEF code generation** for efficient simulation is not properly a verification process; however, the same representation has been used in the Figure 4. The reason is that it requires PN structural

analysis to compute the structural bounds of each place. Once again, the method is scalable because it is possible to easily compute the structural bound of each place of a large component using the previously computed bounds of subcomponents.

Compilation of large scale models is a challenging problem that requires a large amount of computational resources (time and memory). If the size of the model does not fit on a computer, the model is partitioned for the compilation of separated components. HLPN provides higher-level primitives/components defining entities that can be the base to provide a first partition. Model partitions for compilation are guided by component reusability and component size manageability criteria taking into account the computational resources (available memory) for their compilation. It is possible to separately compile manageable-sized modules, which can later be linked when loading the model into the simbots of the distributed simulator by merging transitions over the corresponding VLEFs.

Deployment and distributed system configuration: *Model Partition* for distributed execution requires, a priori, to identify the *good* parts in which the model is divided. The initial model partition can use criteria similar to those used for compilation, that is, it can use component or entities defined in the conceptual model to provide a first model partition. However, distributed partitions can also be independent of conceptual model entities or compiled components.

Model partition verification (see Figure 4) uses HPN and flat PNs models that contribute to a rich structural information of the system. The partition can be supported by applying structural and behavioural analysis.^{46,61,81,86} In this sense, strategies based on the identification of sequential state machines can be used to extract the maximum concurrency (for example, computing p-semiflows in an incremental way). The operational model (hardware architecture and synchronisation algorithms) must be taken into account.⁷⁴ Due to the NP-hardness nature of these problems, research has been directed to find fast approximate solutions using optimisation techniques.^{87,88}

A fundamental aspect of a TPN distributed code concerns the deployment of compiled components and the definition of connections between PNs deployed in different nodes. Spatially large models require for connections to be specified by a scripting configuration language.⁷⁸ Deployed components can be easily connected by the fusion of LEF-coded transitions that conform the component interface. Our LEF-coded representation of TPNs allows us to drill down the partitions to the maximum level of granularity and maximum concurrency: transitions represent events that can be distributed into different components. The only constraint to be considered is to keep transitions in structural conflicts in the same components to maintain conflict-resolution local, and avoid unnecessary messages.⁶²

Dynamic configuration and load-balancing. In a large computing infrastructure, resources are shared by many users, resulting in a much greater variability in the allocation of computing and communication resources. Even if the best model partition were achieved and deployed, small changes in the hardware configuration or the evolution of the simulation would produce large imbalances.⁴⁸ Load-balancing can be supported by considering LPs as containers of interacting entities that can move between LPs.⁴⁸ The set of partitions or LPs in these approaches is stated before the simulation itself. Once the code of the LP is generated it cannot be reconfigured in runtime. Moreover, these approaches lack of an event model that defines what entities are related and how the chain of events is propagated between them, delegating and locking communications and adjacent relationships in LPs.

LEF-coded model is not locked in *Simbots*, which enables the portability of the model between them. The code carries event network dependencies, which can be used by *Simbots* to redefine neighbourhood relationships facilitating dynamic load balancing.

Execution, data-collection, monitoring, fault detection. The architectural proposal for distributed simulation is based on micro-kernels specialised in an interpreted simulation of the model. Each of these simulation micro-kernels is referred to as *Simbots*. A *Simbot* provides an efficient LEF-coded PN interpreter, simulation services such as conservative/optimistic synchronisation protocols, load-balancing mechanisms, as well as monitoring and result collection.¹⁶ **The verification of the model partition** supports finding the best partition dynamically when the workload is unbalanced. The collection of simulation data is also essential for the **validation of the functional model**, as well as for feeding the operational model to improve performance predictions in future simulations.

In addition to the basic services provided by a *Simbot*, large-scale distributed simulation requires a more complete list of services to support an MDE approach. An essential component is a *Simulation Information System*. Conceptual models, compiled code, and collected information must be supported by a database. Non-behavioural semantic aspects of the conceptual model must be removed from the executable code in order to prioritise efficiency, scalability and load balancing. However, the execution of the flat model and the collection of simulation results, require a translation into the conceptual model semantics to be interpreted by the model users. Compilation and elaboration processes store the correspondence between events in the flat model and the conceptual model in a database.

Load balancing will move LEF-coded transitions that reference other transitions. A distributed *Name Service* is required to provide the location of any transition. A large-scale distributed simulation also requires a *fault tolerance system* to configure replicas and detect faults.

5 | MODULAR DESIGN WITH PN-BASED COMPONENTS FOR DISTRIBUTED SIMULATIONS

By providing communicating processes and resources as basic primitives it is possible to represent a large range of problems in different disciplines such as manufacturing,⁸⁹ software,⁹⁰ cloud,¹² or healthcare⁹¹ systems. In this work the methodology followed by most of the approaches to construct PN models is reused: (1) definition of the functional model, (2) definition of the operational model, and (3) construction of the global model by composition through resource sharing or communication.

The methodology is illustrated by two different examples: a manufacturing system, and a wavefront model. The methodology is applied to the modelling of restricted classes of PNs. It shows how the methodology allows the modelling of complex scalable models that can be translated into flat well-formed LEF-coded PNs for distributed simulation. Due to limitations in the length of the article, intentionally simple examples have been chosen. Despite their simplicity, these examples illustrate the possibility of easily scaling the presented models. The modelling methodology presented imposes constraints, such as defining structurally limited models, which are easy to meet since modelling finite resources naturally leads to this type of constraint. The composition of these PN components results in structurally limited PNs, ensuring that the values of the VLEF values are bounded during the simulation. Runtime errors due to variable overflow are not possible. Without these restrictions, following the proposed PN-based methodology, it is possible to avoid these overflow errors in our simulations. However, it can appear in other simulation languages.

5.1 | RAS: Composition of processes and resources

Figure 5 sketches a production cell where parts are processed. The cell is composed by four machines M1, M2, M3, and M4 with capacity for processing concurrently two parts. M1 shares the working place with M2, and M3 shares a working place with M4. There are three robots R1, R2 and R3 to transport parts. R1 transports parts from the input I1 to load machines M1 and M2, and can unload M3 in output O2. R2 can transport parts between the four machines. Finally, R3 can load parts from input I2 and unload parts from M2 to output O2.

Two different types of parts have to be processed. Type 1 parts are taken from the point I1, processed in machine M1 or M3, then in machine M2, and finally unloaded to output O2. Type 2 parts are loaded at input I2, processed in machine M4, then in machine M3, and finally unloaded to output O2.

Left Figure 6 shows the entities to model the cell and processes. The production plan for type 1 parts is modelled by an instance of *Plan1*, and that for type 2 parts by an instance of *Plan2*. Black transitions and diamond symbols represent interface transitions of plan instances. All transitions in the model are in the interface of components. The names of transitions in the plans are T_i . The load of machine instances is named *M1* and the unload *M2*. The transport of parts by the robot starts with *TR1* and ends with *TR2*. Places whose name begins with P_i model states that can be reached by type i parts. For example, $P1M2$ corresponds to a type 1 part being processed in M2. $P1_0$ and $P2_0$ models respectively

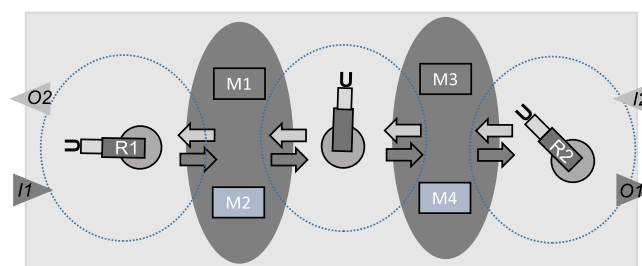


FIGURE 5 Layout of a manufacturing cell.

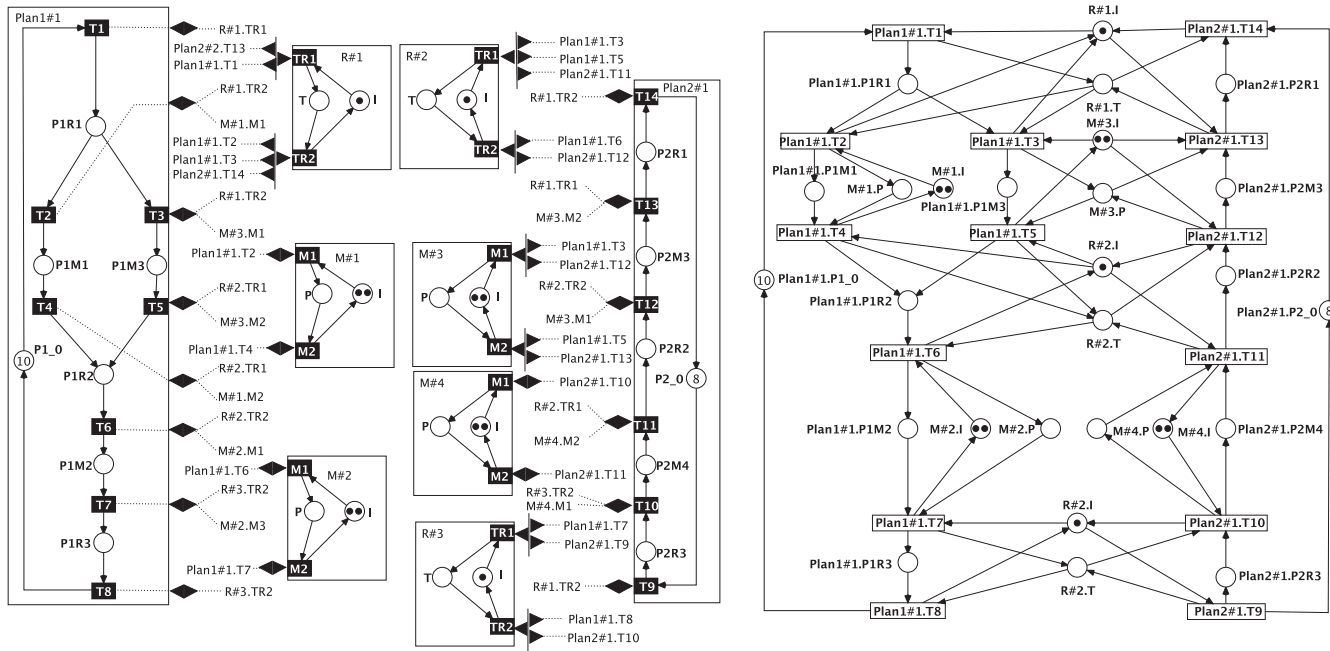


FIGURE 6 RAS perspective in the context of FMS: On the left, entities to model processes (process plans) competing for resources. On the right, resulting flat PN model.

type 1 and 2 parts out of the systems. The initial marking in these places provides upper bounds to places, which allow to calculate initial LEF values of transitions.

Each part in these states needs some resources. Resources $R\#i$ represent instances of robots, and resources $M\#i$ instances of machines. Places named I represent idle resources, P places represent the processing of parts on machines, and T places represent robots transporting a part. At state $P1M2$ the part is using machine $M2$, and needs robots $R2$ and $R3$ to load and unload the machine.

To define the RAS model, production plans of parts are composed with the resources through transition fusion. Note that the links between components represent transitions fusions. To avoid crossing links, only the other fusion transition of the link is labelled in each component interface. For example, $T2$ of plan $Plan\#1$ is linked with $R\#1 . TR2$, using the usual dotted notation to represent the transition $TR2$ of $R\#1$, and with $M\#1 . M1$. All these transitions are merged into one.

On the other side, robot transition $R\#1.TR2$ can upload parts in machine $M1$ and $M3$ for processing parts of plan1, and load parts from $M3$. This means that a replica of transition $TR2$ is merged for each one of these transitions. In order to represent this fact, the diamond symbol that stands for the interface has been split. Therefore, in addition to transition merging, a unary transition *replica* operator is required. This operator, like the merger of transitions, maintains the upper bound of the marking of component places.

Right Figure 6 shows the resulting flat PN obtained after component composition. This model is well defined and belongs to the class called S^4PR . Initially there is no activity in the system, and each possible production path has enough resources to be executed separately. All transitions are live, resources are used in a conservative way, and all places are bounded.³³ This means that once transformed into a flat LEF-coded PN, the simulation is free of deadlocks, and the LEF-values are free of overflows.

Instead of defining the model by composing resources and process plans, we could have defined the cell model in Figure 5 through composition. This simple cell can be instantiated and composed with other cells, transportation systems, warehouses, and so on. Finally, the manufacturing plant model can be composed with process plans. Ultimately, the methodology is easily scalable, allowing for the modelling of complete manufacturing plants and complex process plans.

5.2 | Channels: Composition of processes by asynchronous messages

Communication is the alternative form of interaction to resource sharing between processes. Process interactions imply the use of a *channel* that represents the synchronisation of the internal behaviour of entities. Transition fusion of different

components means that both processes engage simultaneously in the event represented by fusion transition. If buffering is required on a channel, this is achieved by interposing a buffer between the two processes. If the buffer has a limited capacity, the channel can be modelled as a resource shared by sender and receiver. In this case, however, it is an open loop, and there is no competition for the resource.

Typically, the modeller thinks of the communication channel as a buffer of infinite capacity. Left Figure 7 shows two communicating processes and an asynchronous channel. It shows *sender*, *receiver* and the asynchronous channel components, with component interfaces defined as places. In this case, entity interactions are defined by connecting ports as illustrated in the figure. The coupling relation between the output *Sender#1.Mo* and the input port *Channel#1.Mi* results in the fusion of places that represent the delivery of a message to the *Receiver#1* through the channel *Channel#1*, and the connection of *Channel#1.Mo* and *Receiver#1.Mi* port results in the fusion place that shows the message is ready to be received. This sample shows how the language also supports the classical port mapping between components, similar to DEVS, VHDL, or TPN Designer. With this design, a repetitive firing of transitions t_1 , t_2 and t_3 can result in an unbounded number of tokens in place *Channel#1.Mi*. An interpreter playing the token game could cause an error due to an overflow of the LEF value (the number of tokens in a place in classical PN interpreters).

Right Figure 7 shows the alternative representation of structural bounded components with the interface defined by transitions, and links representing the fusion of transitions. The channel is provided with a buffer capacity. In this case, every component is live and bounded, and the component composition results in a well-formed PN.

In order to illustrate the model scalability, the matrix-vector multiplication problem is used. It can be modelled in a streaming fashion where data are flowing through different processing tasks that transform them.^{12,36} A wavefront algorithm modelled in streaming is a systolic computation with the same functionality as the stages conformed by systolic cells arranged in a regular form with the nearest neighbour.

The functional PN model of the wavefront algorithm sketched in Figure 9 is constructed in a modular fashion. The basic component primitives needed in this case are: (1) the component to describe a *systolic cell* in a node of the wavefront array in left Figure 8; (2) a component to describe the *data transmission* of the input and output data streams to/from the wavefront array in right Figure 8.

Once a functional model is achieved, it is necessary to know how it will behave in a specific computing infrastructure. For this purpose, a model of the computational resources and infrastructure is created. Then, the functional model

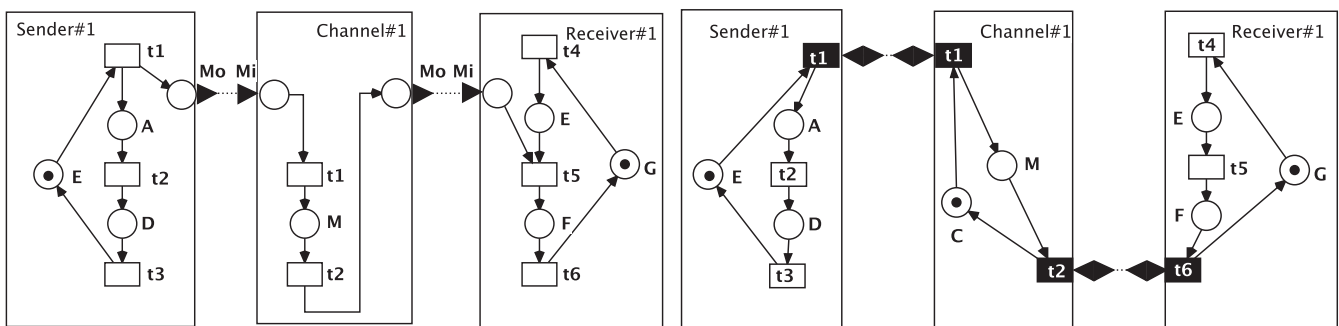


FIGURE 7 Composition based on port mapping and place fusion versus composition based on transition fusion to model processes communicating by a channel. On the left, composition based on port mapping and place fusion. On the right, composition based on transition fusion.

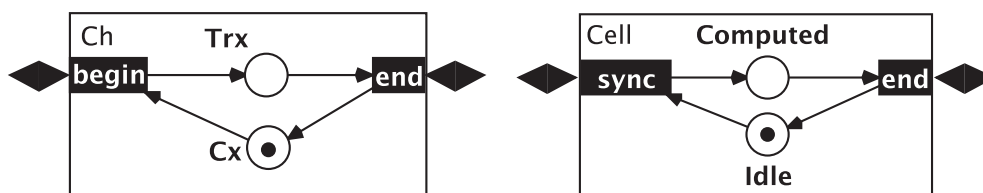


FIGURE 8 Basic modules for the construction of the functional Petri net model of the wavefront algorithm. On the left, computational process associated to a node. On the right, data transmission process for the external/internal data streams.

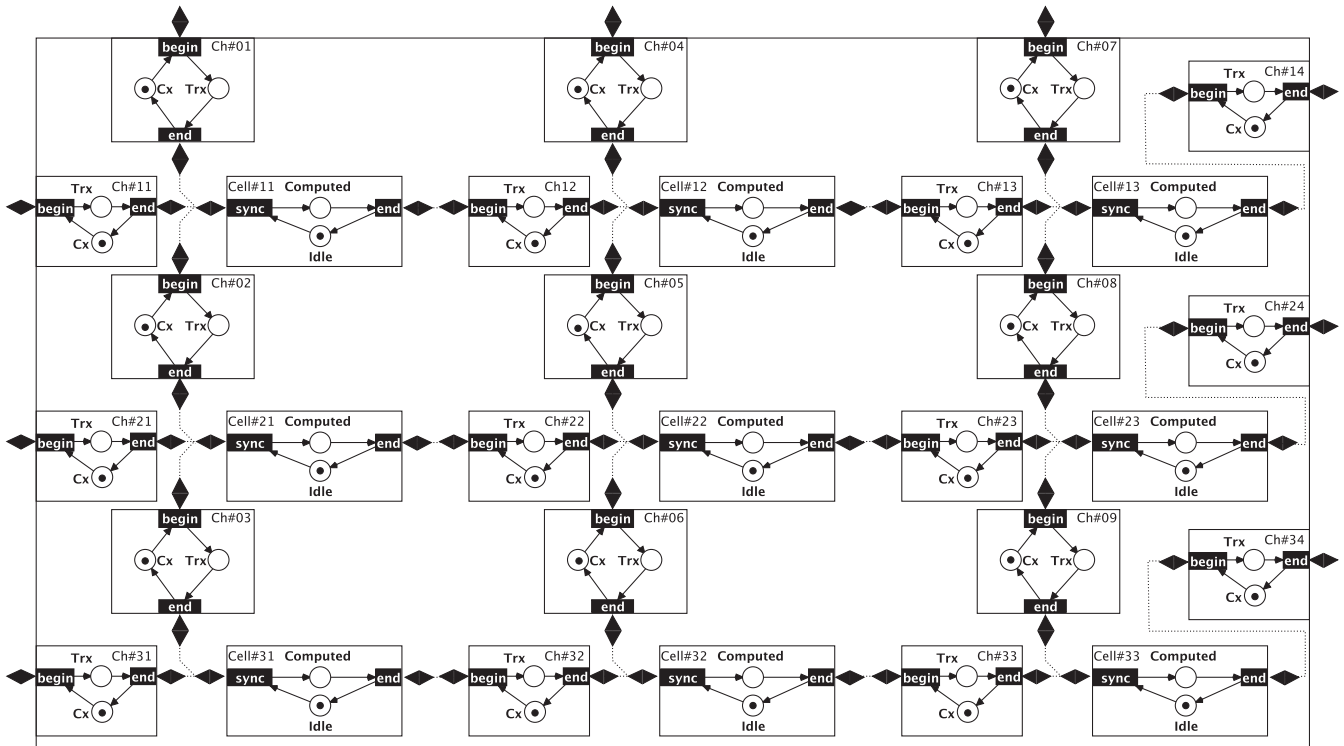


FIGURE 9 Untimed functional Petri net model of the wavefront algorithm.

is combined with the infrastructure model. This is referred to as an operational model. The functional model of the wavefront can be merged with computational resources (e.g., cloud resources) in order to define an operational model and to simulate a wavefront operated in a streaming fashion, or to model the regular systolic cells implemented in hardware. In this example, describing the resources can be as simple as labelling the transitions that represent the usage of resources mentioned at a high level in the functional model with time. Of course, a more comprehensive model would require a more detailed description of the resources and infrastructure. The addition of time to the model of Figure 9 can be done by labelling *begin*'s *Ch#ij* transitions with the expected communication time, and *begin*'s *Cell#ij* transitions with the operation time. In a streaming pipeline in the cloud, performance analysis can be obtained by profiling the cloud infrastructure to feed the current models with time distribution annotations in order to estimate bounds, or by running simulations using probability distributions with different *Coefficients of Variation* in order to estimate the performance on different execution platforms.¹²

The behavioural analysis of the design obtained in Figure 9 can be performed by exploiting the structural properties of the net. It is a *strongly connected marked graph* (a subclass of Petri nets where each place has only one input and one output transition, being strongly connected in the sense of graph theory).¹² The wavefront model has a regular structure that is adequate to generate large-scale models and to test the scalability of a centralised LEF-code based simulator.

6 | EXPERIMENTAL RESULTS

In distributed simulations it is necessary to look for the best system configuration considering latency, bandwidth, model partition, processing rates and memory usage. The point of this article is that a PN-based MDE approach can help in all these aspects related with the M&S of complex systems. The experimental work of the current work is only focused on the feasibility of a LEF based interpreter of TPN models as the first necessary tool for the construction of a framework to support a PN-based MDE approach. By maintaining PNs as the underlying model representation in all stages, it is possible to benefit from a modular PN analysis/simulation throughout the development life-cycle of the M&S process. The LEF-coded interpreter presented in Algorithm 2 has been implemented with version 1.52.1 of the Rust compiler. Rust is a language focused on safety and performance establishing itself as an alternative to C++.

The current experimental evaluation is based on two models with different characteristics. The first model is the wavefront presented in the previous section to illustrate the LEF-based efficiency of the centralised interpreter. This model is easily scalable and presents a high density of events (number of events per unit of simulated time). We will observe that the simulation time behaves well with the scale of the model. When the number of events in the event queue exceeds a certain order of magnitude, the simulator's performance is drastically affected. Managing the event queue is the biggest challenge in the simulation of discrete event systems. The only possible solution is to partition the model and use distributed simulation to reduce the number of events in the event queue. The second model is a synthetic Petri net model that is easily parametrizable, aimed at determining when the workload on the simulator is significant enough to justify distributing the model.

In order to carry out the first experimental task, we used the wavefront model, in which all transitions have been labelled with a unit of time. The interpretation of PNs heavily depends on the model size. The wavefront model presents a high event density, that is, a high number of events by simulated second, and it is greatly affected by the efficiency of event scheduling in the *EVL* queue. Half of the transitions in the model are fired in each simulation time, and each transition firing updates at least two LEF values. It enqueues, for each fired transition, one immediate UF, and a future UF for the transition in the PUL. It supposes that the event queue size is equal to the number of transitions in the model.

Table 1 shows the execution of wavefront models with different number of cells and transitions cells 3×3 , 9×9 , 81×81 , 243×243 and 729×729 , that is, each model has a number of transitions 9 times greater than the previous one. All transitions have a duration of a unit of time, and the simulated time for all experiments is 1,000,000 simulated seconds. The first column shows the number of transitions in the respective models, the second one the number of fired transitions during simulation, the third column the wall-clock time, and the fourth column shows the processing event rate (P) (events/second) that is the usual metric to compare centralised DES simulators. The experimentation has been executed in a PC with an Intel i5-4690 3.50GHz processor, 32 GB of RAM and an Ubuntu 18.04 operating system with kernel version 5.4.0-70-generic.

Figure 10A shows the execution time of the LEF-based interpreter depending on the wavefront model size in a logarithmic scale. As it can be seen, the execution time has a good behaviour until the lack of resources due to the model size slows down the simulation. The event queue, which has been implemented with a binary heap queue, has a size occupancy equal to the number of transitions in the model. It shows a similar behaviour to compiled simulators where the event queue performance is the most relevant aspect. Figure 10B shows the event processing rate based on maximum occupancy of the event queue in a logarithmic scale. It shows the expected behaviour affected by the scheduling of events in the *EVL* in a heap-queue whose cost for operation pop is $O(\log(n))$.

For the second experimental task, we based our work on the previous study of Andras Varga et al.⁹² Authors provide in this work a guide for model partitioning that includes a comprehensive metric considering both the model and the physical characteristics of the node (processing rate and communications) for conservative simulations. They propose that null messages should reach the target LP before it runs out of work, and express it with the condition: $\tau < LHVT/R$, where τ is the latency, $LHVT$ is the local horizon virtual time presented in Section 3.1 (Section 3.1.2), and R is the simulation time advanced rate that shows how many seconds of simulation are advanced by walk-clock seconds. A more intuitive definition of this condition can be expressed as $\tau * P < LHVT * E$, where P is the event processing rate (events/sec), and E is the event density, that is, the number of processed events per simulated second in (events/simSec) that can be calculated as $E = P/R$. This expression shows that the number of events the CPU can process during message transit ($\tau * P$), which are dependent on CPU, network latency, and interpreter efficiency, must be less than the number of events to

TABLE 1 Centralised LEF-based simulation of wavefront model with different model sizes.

Transitions	Fired	Time (s)	P (ev/s)
21	10,999,972	0.31	35957609.14
171	86,998,710	2.76	31552801.68
1485	746,961,948	32.47	23001560.58
13,203	6,613,948,350	429,27	15407567.28
118,341	59,182,401,492	9586.95	6173226.22
1,063,611	531,153,047,430	117970.90	4502407.39

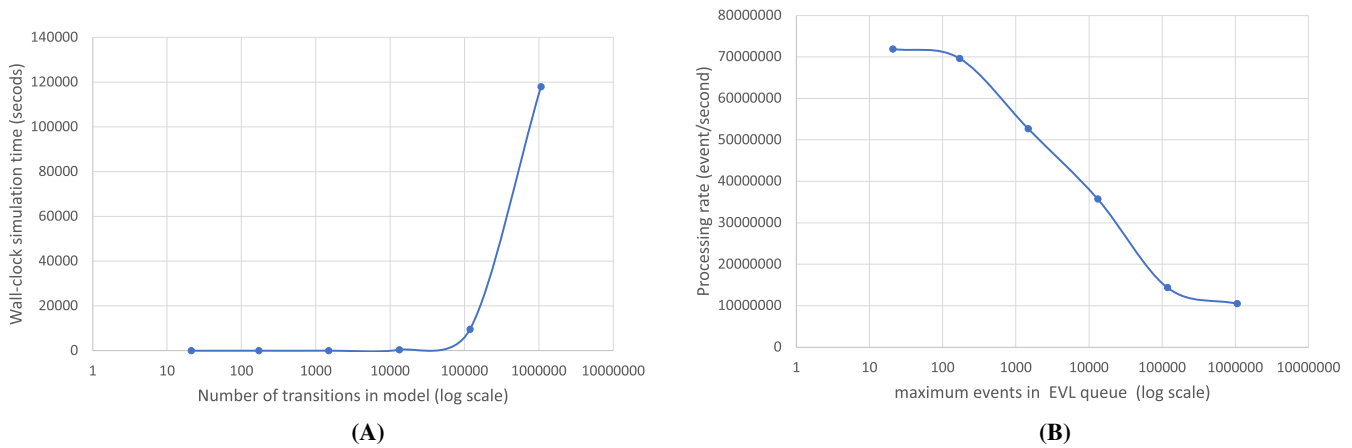


FIGURE 10 Walk-clock simulation time and event processing rate of centralised LEF-based simulation with different wavefront model sizes. (A) Walk-clock simulation time in function model size. (B) Event processing rate in function of max event queue size.

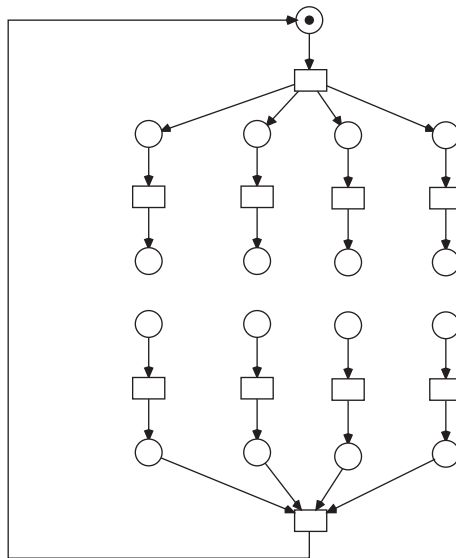


FIGURE 11 Model to configure Petri nets with different sizes/workload for distributed simulation.

be processed by the LP until the next horizon time defined by the *LBTS*, which is model dependent. A more detailed discussion about the work load in distributed simulation can be found in Reference 15.

The model in Figure 11 has been used for supporting the experimentation of distributed simulations. This synthetic PN can be easily parametrised with the number of branches, which are chains of events that can be executed in parallel without violating the causality constraint, and the number of transitions that represents the simulation workload of each branch. This model has a low event density (equal to one for each branch). In this way, the simulation is not affected by the event management efficiency of the *EVL*. The model can be easily partitioned and scaled with the number of branches and their length. A greater branch length implies a higher lookahead value. The distributed simulation has been implemented following the LEF-PN code interpreted by Algorithm 2, and the conservative distributed simulation Algorithm 1 presented in Section 3.

It is important to note that the presented models are defined without adding any synthetic load associated to the occurrence of events. The consideration of this additional load in each node is very relevant and could require the processing of large amount of data. It is an additional aspect to be considered for scalability issues. In the current case, the model examples are presented without adding any synthetic load, and the data processing is simulated with transition annotation of the estimated load processing time. Scalability issues related with data processing have been considered orthogonal to distributed simulation. They have their own scalability and load balancing mechanisms.

TABLE 2 Distributed simulation versus centralised simulation with different loads by *Simbot* implemented in rust.

Simulation	#br.	trans/br.	Events	Nodes	Events/s	Exec. time
Centr.	2	10,000	19,998,003	1	7,522,936	2.658 s
Distr.	2	10,000	19,998,003	3	2,609,886	7.619 s
Centr.	7	10,000	69,988,013	1	4,407,922	17.79 s
Distr.	7	10,000	69,988,013	8	6,652,821	10.52 s

This decision leads the disadvantage of distributed setups with respect to the centralised approach, from a performance evaluation point of view. Distributing the model means parallel computation, but involves event interchanges subject to bandwidth and latency constraints.⁹³

Table 2 shows the results obtained from the conservative simulator implemented in Rust language running on a mini cluster of Raspberries Pi 4 with 8 GB of RAM each. It also illustrates the advantage of using a simulation micro-kernel, which allows to deploy the model partitions on different hardware. The same PNs, modelled as Figure 9, have been tested with variations in depth and number of branches. The simulated time for these experiments is 10M SimSeconds.

First two rows show a lower load case: 2 branches with 10K transitions per branch. It is found that better results are obtained for the centralised simulator. In the next two rows, the number of branches has been increased, trying to find a situation that is more suitable to the use of a distributed simulator. In this case, the real simulation time is less in the distributed version, which indicates that if the number of branches that run in parallel increases, the simulation time of the centralised one substantially does too, so that a distributed solution is more suitable. The sixth column shows the total number of events/second in the simulation. The number of nodes is one by branch, and one additional node contains the PN part that synchronises all branches at the beginning and at the end. This *Simbot* contains a negligible number of transitions and is not considered for the processing rate of each branch. The number of events/second reached in the last case with 7 branches is approximately 6,652,821, which is near the centralised approach in the first row but executing a larger model.

7 | CONCLUSIONS AND FUTURE WORK

This article focuses on the role of PNs as the core formalisms to support an MDE approach for the M&S of large models. PNs are an executable specification, which make them particularly suitable for M&S of DES. However, formal-model based analysis tools are only useful under certain assumptions or are insufficient to afford the study of large models.

This work reviews the role of hierarchical PNs in each stage of the M&S of large DES, from conceptual modelling to the generation of code execution. From the modelling point of view, it has been proven that it is possible to ensure well-formedness composition of PNs using structurally constrained PNs adequate for the domain. In this way, large-scale PN models deadlock free and bounded can be obtained. From the simulation point of view, the paper presents a PN model adequate for efficient interpretation and configuration of the simulation. LEF-coded PN components can be composed by transition fusion, which preserves bounded models. Separate compilation and composition allow the development of large-scale models.

P/T nets explicitly specify preconditions and state changes. This offers the possibility of compiling them in an event-dependency network. The interest of LEFs lies in their use in the simulation of PNs to reduce the number of tests (to determine the enabling of a transition) during the simulation cycle, with respect to a place-driven simulation. The proposed LEF-code structure maintains the enabling-transition state when transitions fire. It uses the event-dependency network to update the function value of affected transitions and requires only integer addition operations.

From the perspective of distributed simulation, the desirable characteristics of the code are the facilities to scale and adapt to the execution environment through mechanisms such as the dynamic composition of components, and load balancing. The main contributions of this work from this point of view are:

- The exchange of messages between simulators to update the enabling transition state is reduced to the transmission of integers. It is the minimum information to be transmitted from a transition that has occurred to another transition when the marking contents of their input places have been modified. Therefore, the costs of transmitting messages through the communication network are reduced to the essential minimum.

- The compilation of large-scale models is an important bottleneck for the generation of executable code and the deployment in large scale execution infrastructures. VLEFs provide the adequate transition characterisation to support the separated compilation of modules or parts of the model, and component linkage of distributed simulators.
- PNs can be dynamically composed at simulation time when transitions involved in the fusion are assigned to the same simulation engine by stacking the data structures of VLEFs.
- LEFS present good characteristics for load balancing: (1) the model is not wired with the simulator, which enables its portability to other simulators, (2) the representation of the event-dependency network facilitates the evolution of a dynamic *graph of LPs* since the migration of code carries the event-network dependencies that can be used by interpreters to redefine neighbourhood relationships, and (3) the granularity of the code is not defined at compiled time, and it can be used at different levels of abstraction (from large components down to transitions).¹⁵

The LEF-based dependency-network representation of PNs supposes a new perspective of PNs, which opens many lines of future work. The main objective is to develop a framework for the definition, management, and distributed simulation of large-scale PN models. It comprises HLPN specifications, elaboration, analysis, partitioning, compilation, management, deployment and monitoring.

Current efforts are directed at the implementation of a scalable version of the compiler based on the VLEF structures, and the exploitation of the PN structure for the simulation of scalable systems.

AUTHOR CONTRIBUTIONS

Unai Arronategui: Conceptualization; methodology; software development; formal analysis; investigation; writing – original draft; funding acquisition. **José Ángel Bañares:** Conceptualization; methodology; software development; formal analysis; investigation; writing – original draft; funding acquisition. **José Manuel Colom:** Conceptualization; methodology; software development; formal analysis; investigation; writing – original draft; funding acquisition.

ACKNOWLEDGEMENTS

This work was co-financed by the Aragonese Government and the European Regional Development Fund ‘Construyendo Europa desde Aragón’ (COSMOS research group, ref. T35_17D); and by the Spanish program ‘Programa Estatal del Generación de Conocimiento y Fortalecimiento Científico y Tecnológico del Sistema de I+D+i’, project PG2018-099815-B-100.

CONFLICT OF INTEREST STATEMENT


The authors declare no potential conflict of interests.

DATA AVAILABILITY STATEMENT

Data sharing is not applicable to this article as no new data were created or analysed in this study.

ORCID

Unai Arronategui  <https://orcid.org/0000-0003-4457-3938>

José Ángel Bañares  <https://orcid.org/0000-0002-4198-8241>

José Manuel Colom  <https://orcid.org/0000-0001-5066-4030>

REFERENCES

1. Arronategui U, Bañares JÁ, Colom JM. A MDE approach for modelling and distributed simulation of health systems. *GECON 2020 - International Conference on the Economics of Grids, Clouds, Systems, and Services*. Springer; 2020:89-103.
2. Mahulea C, Mahulea L, García-Soriano J-M, Colom J-M. Petri nets with resources for modeling primary healthcare systems. *2014 18th International Conference on System Theory, Control and Computing (ICSTCC)*. IEEE; 2014:639-644. doi:10.1109/ICSTCC.2014.6982489
3. Zhou J, Reniers G. Petri-net based simulation analysis for emergency response to multiple simultaneous large-scale fires. *J Loss Prev Process Ind*. 2016;40:554-562.
4. Bosmans S, Mercelis S, Hellinckx P, Denil J. Towards evaluating emergent behavior of the Internet of Things using large scale simulation techniques (wip). *Proceedings of the Theory of Modeling and Simulation Symposium, TMS'18*. Society for Computer Simulation International; 2018:1-8.
5. Sood V, Nema MK, Kumar R, Nene MJ. A framework for prototyping distributed cyber–physical systems with reference nets. *Simul Model Pract Theory*. 2022;117:102488.

6. Latorre-Biel J-I, Faulín J, Juan AA, Jiménez-Macías E. Petri net model of a smart factory in the frame of industry 4.0. *IFAC Pap OnLine*. 2018;51(2):266-271.
7. Recalde L, Suárez MS, Ezpeleta J, Teruel E. Petri nets and manufacturing systems: an examples-driven tour. In: Desel J, Reisig W, Rozenberg G, eds. *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*. Lecture Notes in Computer Science. Vol 3098. Springer; 2003:742-788.
8. Carbone A, Ajmone-Marsan M, Axhausen KW, Batty M, Masera M, Rome E. Complexity aided design. *Eur Phys J Spec Top*. 2012;214(1):435-459.
9. Rajhans A, Cheng S-W, Schmerl B, et al. An architectural approach to the design and analysis of cyber-physical systems. Third International Workshop on Multi-Paradigm Modeling; 2009.
10. Zeigler BP, Praehofer H, Kim TG. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press; 2000.
11. Topçu O, Durak U, Oğuztüzün H, Yılmaz L. Distributed simulation: a model-driven engineering approach. *Simulation Foundations, Methods and Applications*. Springer International Publishing; 2016.
12. Tolosana-Calasanç R, Bañares JÁ, Colom J-M. Model-driven development of data intensive applications over cloud resources. *Future Gener Comput Syst*. 2018;87:888-909.
13. Muro-Medrano PR, Bañares JA, Villarroel JL. Knowledge representation-oriented nets for discrete event system applications. *IEEE Trans Syst Man Cybern A Syst Hum*. 1998;28(2):183-198.
14. Arronategui U, Bañares JÁ, Colom JM. Towards an architecture proposal for federation of distributed DES simulators. *GECON 2019 - International Conference on the Economics of Grids, Clouds, Systems, and Services*. Lecture Notes in Computer Science. Vol 11819. Springer; 2019:97-110.
15. Hodgetts P, Kocharyan H, Reviriego F, et al. Workload evaluation in distributed simulation of DESs. *GECON 2021 - International Conference on the Economics of Grids, Clouds, Systems, and Services*. Lecture Notes in Computer Science. Vol 13072. Springer; 2021:3-16.
16. Bañares JÁ, Colom JM. Model and simulation engines for distributed simulation of discrete event systems. *GECON 2018 - International Conference on the Economics of Grids, Clouds, Systems, and Services*. Lecture Notes in Computer Science. Vol 11113. Springer; 2018:77-91.
17. SISO, SAC. *IEEE Standard for Modeling and Simulation High Level Architecture (HLA)—Framework and Rules*. IEEE Std. Std 1516-2010; 2010:1-38.
18. Boukerche A, Grande RED. Optimized federate migration for large-scale HLA-based simulations. *2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*. IEEE; 2008:227-235.
19. Cassandras CG, Lafortune S. *Introduction to Discrete Event Systems*. 3rd ed. Springer Publishing Company; 2021.
20. Seatzu C, Silva M, van Schuppen JH. *Control of Discrete-Event Systems: Automata and Petri Net Perspectives*. Springer Publishing Company; 2012.
21. Baier C, Katoen J. *Principles of Model Checking*. MIT Press; 2008.
22. Cederbladh J, Cicchetti A, Suryadevara J. Early validation and verification of system behaviour in model-based systems engineering: a systematic literature review. *ACM Trans Softw Eng Methodol*. 2024;33(3):81.
23. Eshuis R. Statechartable Petri nets. *Form Asp Comput*. 2013;25(5):659-681.
24. Grammes R, Gotzhein R. SDL profiles: formal semantics and tool support. *Proceedings of the 10th International Conference on Fundamental Approaches to Software Engineering*. Springer-Verlag; 2007:200-214.
25. Fonseca i Casas P, Lijia Hu D, Guasch i Petit A, Figueras i Jové J. Simplifying the verification of simulation models through Petri net to FlexSim mapping. *Appl Sci*. 2020;10(4):1395.
26. Fonseca i Casas, P, Martin, C. Building models in pairs for cross-verification using SDL and DEVS. *Adv Theory Simul*. 2024;7:2300839.
27. Bause F, Kabutz H, Kemper P, Kritzinger PS. SDL and Petri net performance analysis of communicating systems. In: Dembinski P, Sredniawa M, eds. *Protocol Specification, Testing and Verification XV, Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification, Warsaw, Poland, June 1995, IFIP Conference Proceedings*. Vol 38. Chapman & Hall; 1995:269-282.
28. Boukelkoul S, Redjimi M. Mapping between Petri nets and DEVS models. *2013 3rd International Conference on Information Technology and e-Services (ICITeS)*. IEEE; 2013:1-6.
29. Moreno RP, Salcedo JLV. Evaluation of the ability of Petri net centralized implementation techniques. *50th IEEE Conference on Decision and Control and European Control Conference, 11th European Control Conference, CDC/ECC 2011, Orlando, FL, USA, December 12–15, 2011*. IEEE; 2011:403-410.
30. Colom JM. Harnessing structure theory of Petri nets in discrete event system simulation. In: Kristensen LM, van der Werf JMEM, eds. *Application and Theory of Petri Nets and Concurrency - 45th International Conference, PETRI NETS 2024, Geneva, Switzerland, June 26–28, 2024, Proceedings, Lecture Notes in Computer Science*. Vol 14628. Springer; 2024:3-23.
31. Fujimoto RM. Research challenges in parallel and distributed simulation. *ACM Trans Model Comput Simul*. 2016;26(4):22:1-22:29.
32. Dori D, Bolshchikov S, Wengrowicz N. Conceptual models become alive with vivid OPM: how can animated visualization render abstract ideas concrete? In: Gianni D, D'Ambrogio A, Tolk A, eds. *Modeling and Simulation-Based Systems Engineering Handbook*. CRC Press; 2014:293-320.
33. Colom J. The resource allocation problem in flexible manufacturing systems. In: Best E, Aalst W, eds. *Application and Theory of Petri Nets 2003*. Lecture Notes in Computer Science. Vol 2679. Springer-Verlag; 2003:23-35.
34. van der Aalst WMP, van Hee KM. *Workflow Management: Models, Methods, and Systems*. Cooperative Information Systems. MIT Press; 2002.
35. Reisig W. Deterministic buffer synchronization of sequential processes. *Acta Inform*. 1982;18:117-134.

36. Ferscha A. A Petri net approach for performance oriented parallel program design. *J Parallel Distrib Comput*. 1992;15(3):188-206.
37. Ramirez A, Campos J, Silva M. On optimal scheduling in DEDS. *Proceedings IEEE International Conference on Robotics and Automation*. Vol 3. IEEE; 1993:821-826.
38. Merino A, Tolosana-Calasanz R, Bañares JÁ, Colom J-M. A specification language for performance and economical analysis of short term data intensive energy management services. *Economics of Grids, Clouds, Systems, and Services*. Springer; 2016:147-163.
39. Traoré MK, Zacharewicz G, Duboz R, Zeigler BP. Modeling and simulation framework for value-based healthcare systems. *Simulation*. 2019;95(6):481-497.
40. Gomes L, Barros JP. Structuring and composability issues in Petri nets modeling. *IEEE Trans Industr Inform*. 2005;1(2):112-123.
41. Jensen K, Kristensen LM, Wells L. Coloured Petri nets and CPN tools for modelling and validation of concurrent systems. *Int J Softw Tools Technol Transf*. 2007;9(3-4):213-254.
42. Kummer O, Wienberg F, Duvigneau M, et al. An extensible editor and simulation engine for Petri nets: renew. In: Cortadella J, Reisig W, eds. *Applications and Theory of Petri Nets 2004, 25th International Conference, ICATPN 2004, Bologna, Italy, June 21-25, 2004, Proceedings*. Lecture Notes in Computer Science. Vol 3099. Springer; 2004:484-493.
43. Desel J, Esparza J. Free choice Petri nets; 1995.
44. Recalde L, Teruel E, Silva M. Structure theory of multi level deterministically synchronized sequential processes. *Theor Comput Sci*. 2001;254:1-33.
45. López-Grao J, Colom JM. A Petri net perspective on the resource allocation problem in software engineering. *Transactions on Petri Nets and Other Models of Concurrency V*. Vol 6900. Springer; 2012:181-200.
46. Moreno RP, Tardioli D, Salcedo JLV. Distributed implementation of discrete event control systems based on Petri nets. *2008 IEEE International Symposium on Industrial Electronics*. IEEE; 2008:1738-1745.
47. Piedrafita R, Villarroel JL. Performance evaluation of Petri nets centralized implementation. The execution time controller. *Discret Event Dyn Syst*. 2011;21(2):139-169.
48. DflAngelo G. The simulation model partitioning problem: an adaptive solution based on self-clustering. *Simul Model Pract Theory*. 2017;70:1-20.
49. Perumalla KS. μ sik - a micro-kernel for parallel/distributed simulation systems. *Workshop on Principles of Advanced and Distributed Simulation (PADS'05)*. IEEE Computer Society; 2005:59-68.
50. Bruno G, Elia A. Operational specification of process control systems: execution of PROT nets using OPS5. In: Kugler H, ed. *Information Processing 86, Proceedings of the IFIP 10th World Computer Congress, Dublin, Ireland, September 1-5, 1986*. North-Holland/IFIP; 1986:35-40.
51. Colom J, Silva M, Villarroel J. On software implementation of Petri nets and colored Petri nets using high-level concurrent languages. *7th International Workshop on Application and Theory of Petri Nets*; 1986:207-222.
52. Silva M, David R. Synthèse programmée des automatismes logiques décrits par réseaux de Petri: Une méthode de mise en oeuvre sur microcalculateurs; 1979:369-393.
53. Silva M, Velilla S. Programmable logic controllers and Petri nets: a comparative study. In: Ferrate G, Puente E, eds. *Software for Computer Control 1982*. Pergamon; 1983:83-88.
54. Valette R. Nets in production systems. In: Brauer W, Reisig W, Rozenberg G, eds. *Petri Nets: Central Models and their Properties, Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course, Bad Honnef, Germany, 8-19 September 1986*. Lecture Notes in Computer Science. Springer; 1986:191-217.
55. Briz JL, Colom JM. Implementation of weighted place/transition nets based on linear enabling functions. *International Conference on Application and Theory of Petri Nets*. Springer; 1994:99-118.
56. Briz J, Colom J, Silva M. Simulation of Petri nets and linear enabling functions. *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*. Vol 2. IEEE; 1994:1671-1676.
57. Gaeta R. Efficient discrete-event simulation of colored Petri nets. *IEEE Trans Softw Eng*. 1996;22(9):629-639.
58. Mäkelä M. Optimising enabling tests and unfoldings of algebraic system nets. In: Colom JM, Koutny M, eds. *Application and Theory of Petri Nets 2001, 22nd International Conference, ICATPN 2001, Newcastle upon Tyne, UK, June 25-29, 2001, Proceedings*. Lecture Notes in Computer Science. Springer; 2001:283-302.
59. Valette R, Bako B. Software implementation of Petri nets and compilation of rule-based systems. In: Rozenberg G, ed. *Advances in Petri Nets 1991, Papers from the 11th International Conference on Applications and Theory of Petri Nets, Paris, France, June 1990*. Lecture Notes in Computer Science. Vol 524. Springer; 1990:296-316.
60. Olcoz S, Colom JM. VHDL: a discrete event simulation hardware description language. *Proceedings of 1994 International Conference on Simulation and Hardware Description Languages (SHDL'94)*; 1994:128-134.
61. Ammar HH, Deng S. Parallel simulation of stochastic Petri nets using spatial decomposition. *Proceedings of the 1991 IEEE International Symposium on Circuits and Systems*. Vol 2. IEEE; 1991:826-829.
62. Chiola G, Ferscha A. Distributed simulation of Petri nets. *IEEE Parallel Distrib Technol Syst Appl*. 1993;1(3):33-50.
63. Cicirelli F, Furfaro A, Nigro L. Distributed simulation of modular time Petri nets: an approach and a case study exploiting temporal uncertainty. *Real Time Syst*. 2007;35(2):153-179.
64. Djemame K, Gilles DC, Mackenzie LM, Bettaz M. Performance comparison of high-level algebraic nets distributed simulation protocols. *J Syst Archit*. 1998;44(6-7):457-472.
65. Nicol DM, Mao W. Automated parallelization of timed Petri-net simulations. *J Parallel Distrib Comput*. 1995;29(1):60-74.
66. Thomas GS, Zahorjan J. Parallel simulation of performance Petri nets: extending the domain of parallel simulation. *1991 Winter Simulation Conference Proceedings*. IEEE; 1991:564-573.

67. Best E, Darondeau P. Petri net distributability. In: Clarke E, Virbitskaite I, Voronkov A, eds. *Perspectives of Systems Informatics*. Springer; 2012:1-18.
68. Harel D, Pnueli A. On the development of reactive systems. In: Apt KR, ed. *Logics and Models of Concurrent Systems*. Springer; 1985:477-498.
69. Pnueli A. Specification and development of reactive systems. In: Kugler H, ed. *Information Processing 86, Proceedings of the IFIP 10th World Computer Congress, Dublin, Ireland, September 1-5, 1986*. North-Holland/IFIP; 1986:845-858.
70. Störrle H. *Models of Software Architecture: Design and Analysis With UML and Petri Nets*. Ph.D. thesis. Ludwig Maximilian University of Munich; 2000.
71. Fujimoto R, Bock C, Chen W, Page E, Panchal JH. *Research Challenges in Modeling and Simulation for Engineering Complex Systems*. 1st ed. Springer Publishing Company; 2017.
72. Marsan MA, Balbo G, Bobbio A, Chiola G, Conte G, Cumani A. The effect of execution policies on the semantics and analysis of stochastic Petri nets. *IEEE Trans Softw Eng*. 1989;15(7):832-846.
73. Schriber TJ, Brunner DT, Smith JS. How discrete-event simulation software works and why it matters. *Proceedings of the Winter Simulation Conference, WSC'12*. IEEE; 2012:3:1-3:15.
74. D'Angelo G, Marzolla M. New trends in parallel and distributed simulation: from many-cores to cloud computing. *Simul Model Pract Theory*. 2014;49:320-335.
75. De Grande RE, Boukerche A. Dynamic balancing of communication and computation load for HLA-based simulations on large-scale distributed systems. *J Parallel Distrib Comput*. 2011;71(1):40-52.
76. Kramer J, Magee J. Dynamic configuration for distributed systems. *IEEE Trans Softw Eng*. 1985;11(4):424-436.
77. Brogi A, Canciani A, Soldani J, Wang P. Modelling the behaviour of management operations in cloud-based applications. In: Moldt D, Rölke H, Störrle H, eds. *Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE'15), CEUR Workshop Proceedings*. Vol 1372. CEUR-WS.org; 2015:191-205.
78. Carullo L, Furfaro A, Nigro L, Pupo F. Modelling and simulation of complex systems using TPN designer. *Simul Model Pract Theory*. 2003;11(7):503-532.
79. Silva M, Teruel E, Colom JM. *Linear Algebraic and Linear Programming Techniques for the Analysis of Place/Transition Net Systems*. Springer; 1998:309-373.
80. Notomi M, Murata T. Hierarchical reachability graph of bounded Petri nets for concurrent-software analysis. *IEEE Trans Softw Eng*. 1994;20(5):325-336.
81. Ferscha A. Parallel and distributed simulation of discrete event systems. *Handbook of Parallel and Distributed Computing*. McGraw-Hill; 1995:1003-1041.
82. Fujimoto RM, Bagrodia R, Bryant RE, et al. Parallel discrete event simulation: the making of a field. *2017 Winter Simulation Conference (WSC)*. IEEE; 2017:262-291.
83. Fujimoto RM. Parallel discrete event simulation. *Commun ACM*. 1990;33(10):30-53.
84. Gracia VM, Tolosana-Calasanz R, Bañares JÁ, Arronategui U, Rana OF. Characterising resource management performance in Kubernetes. *Comput Electr Eng*. 2018;68:286-297.
85. Jr JG, Sullivan F. Issues in event analysis for recursive simulation. *Proceedings of the 37th Winter Simulation Conference, Orlando, FL*. IEEE Computer Society; 2005:1234-1241.
86. García F, Villarroel J. Decentralized implementation of real-time systems using time Petri nets. Application to mobile robot control. *IFAC Proc Vol*. 1998;31(4):11-16.
87. Boukerche A. An adaptive partitioning algorithm for distributed discrete event simulation systems. *J Parallel Distrib Comput*. 2002;62(9):1454-1475.
88. Meraji S, Tropper C. Optimizing techniques for parallel digital logic simulation. *IEEE Trans Parallel Distrib Syst*. 2012;23(6):1135-1146.
89. Ezpeleta J, Colom JM, Martínez J. A Petri net based deadlock prevention policy for flexible manufacturing systems. *IEEE Trans Rob Autom*. 1995;11(2):173-184.
90. Tricas F, Colom JM, Merelo Guervós JJ. Computing minimal siphons in Petri net models of resource allocation systems: an evolutionary approach. *Proceedings of the International Workshop on Petri Nets and Software Engineering, Tunis, Tunisia*; 2014; 1160:307-322.
91. Mahulea C, Garcia-Soriano J, Colom JM. Modular Petri net modeling of the Spanish health system. *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation, ETFA 2012, Krakow, Poland, September 17-21, 2012*. IEEE; 2012:1-8.
92. Varga A, Sekercioglu Y, Egan G. A practical efficiency criterion for the null message algorithm. In: Verbraeck A, Hlupic V, eds. *Simulation in Industry: Proceedings of the 15th European Simulation Symposium (ESS 2003)*. SCS Europe Publishing House; 2003:81-92.
93. D'Angelo G, Ferretti S. Adaptive parallel and distributed simulation of complex networks. *J Parallel Distrib Comput*. 2022;163:30-44.

How to cite this article: Arronategui U, Bañares JÁ, Colom JM. Large scale system design aided by modelling and DES simulation: A Petri net approach. *Softw: Pract Exper*. 2024;1-29. doi: 10.1002/spe.3374