

PIPE2: A Tool for the Performance Evaluation of Generalised Stochastic Petri Nets

Nicholas J. Dingle William J. Knottenbelt Tamas Suto

Department of Computing, Imperial College London,
180 Queen's Gate, London SW7 2BZ, United Kingdom
{njd200,wjk,suto}@doc.ic.ac.uk

ABSTRACT

This paper presents an overview of Platform-Independent Petri Net Editor 2 (*PIPE2*), an open-source tool that supports the design and analysis of Generalised Stochastic Petri Net (GSPN) models. *PIPE2*'s extensible design enables developers to add functionality via pluggable analysis modules. It also acts as a front-end for a parallel and distributed performance evaluation environment. With *PIPE2*, users are able to design and evaluate performance queries expressed in the Performance Tree formalism.

Keywords

PIPE2, Performance Trees, GSPNs, Stochastic Modelling, Parallel and Distributed Computing

1. INTRODUCTION

Platform-Independent Petri Net Editor 2 (*PIPE2*) [1] is a Java-based tool for the construction and analysis of Generalised Stochastic Petri Net (GSPN) [2] models. *PIPE2* began life in 2002/3 as a postgraduate team programming project in the Department of Computing at Imperial College London and has been steadily improved through a number of successive successive versions, implemented by students at Imperial College London and with input from industry (particularly Intelligent Automation, Inc.) In addition, a branched version with significant improvements to different aspects of functionality (e.g. the addition of inhibitor arcs and fixed-capacity places, and an experimental framework) has also been implemented at the Universitat de les Illes Balears [3, 4]. It is planned to merge this functionality into the main branch in due course.

Besides standard Petri net model creation, manipulation and animation facilities, *PIPE2* provides a mechanism for the run-time integration of new functionality via pluggable analysis modules. This is a feature that sets *PIPE2* apart from many other Petri net tools, whose analysis functionality is usually fixed and cannot be augmented by the user. This consequently provides a springboard for experimentation with new analysis techniques without the need to reimplement basic functionality.

The focus of the present paper is on the recent development effort undertaken at Imperial College London related to the Performance Query Editor module, which allows users to create and evaluate performance queries using the Performance Tree (PT) formalism [5–7]. This module links with an evaluation environment comprising an Analysis Server, a

set of parallel and distributed analysis tools, and a hardware cluster that enables large-scale computations.

In terms of impact, *PIPE2* has been used in a number of studies into research topics as varied as the modelling of mobile communication [8] and biological systems [9], and the visualisation of Business Process Execution Language (BPEL) specifications as Petri nets [10]. In addition, a number of other researchers have either implemented their own modules for use with *PIPE2* (e.g. to translate Petri nets into the simulation language SIMAN [11]), or have used *PIPE2* as the basis for the implementation of their own domain-specific versions (e.g. UIB [4] and Exhost-PIPE [12]). Finally, *PIPE2* has been used as a teaching aid at a number of universities, including the Université de Genève, Sweden's Royal Technical Institute (KTH), the University of Leicester, Friedrich-Alexander-Universität Erlangen-Nürnberg and Universität Duisberg-Essen.

The remainder of this paper is organised as follows: Section 2 presents *PIPE2*'s model design and analysis functionality. Section 3 presents background material on performance query specification with Performance Trees (PTs), before Section 4 describes the architecture of the parallel and distributed PT query evaluation environment. Section 5 describes the process of using *PIPE2* to evaluate a PT performance query, and Section 6 demonstrates a case study analysis of a query on a model of a hospital's Accident and Emergency department. Section 7 concludes and discusses future work.

2. SYSTEM MODELLING WITH PIPE2

Figure 1 shows *PIPE2*'s graphical user interface for the creation and editing of GSPN models. Models are drawn on a canvas using components from a drawing toolbar including places, transitions, arcs and tokens. Nets of arbitrary complexity can be drawn and annotated with additional user information. Besides basic model design functionality, the designer interface provides features such as zoom, export, tabbed editing and animation. The animation mode is particularly useful for aiding users in the intuitive verification of the behaviour of their models. *PIPE2* uses the Petri Net Markup Language (PNML) [13] as its file format, which permits interoperability with other Petri net tools including P3 [14], WoPeD [15] and Woflan [16]

Central to the architecture of *PIPE2* is its support for modules, which allow its functionality to be extended at run-time with user-implemented code. *PIPE2* comes equipped with a number of specialised analysis modules that perform structural and performance-related analyses on GSPN mod-

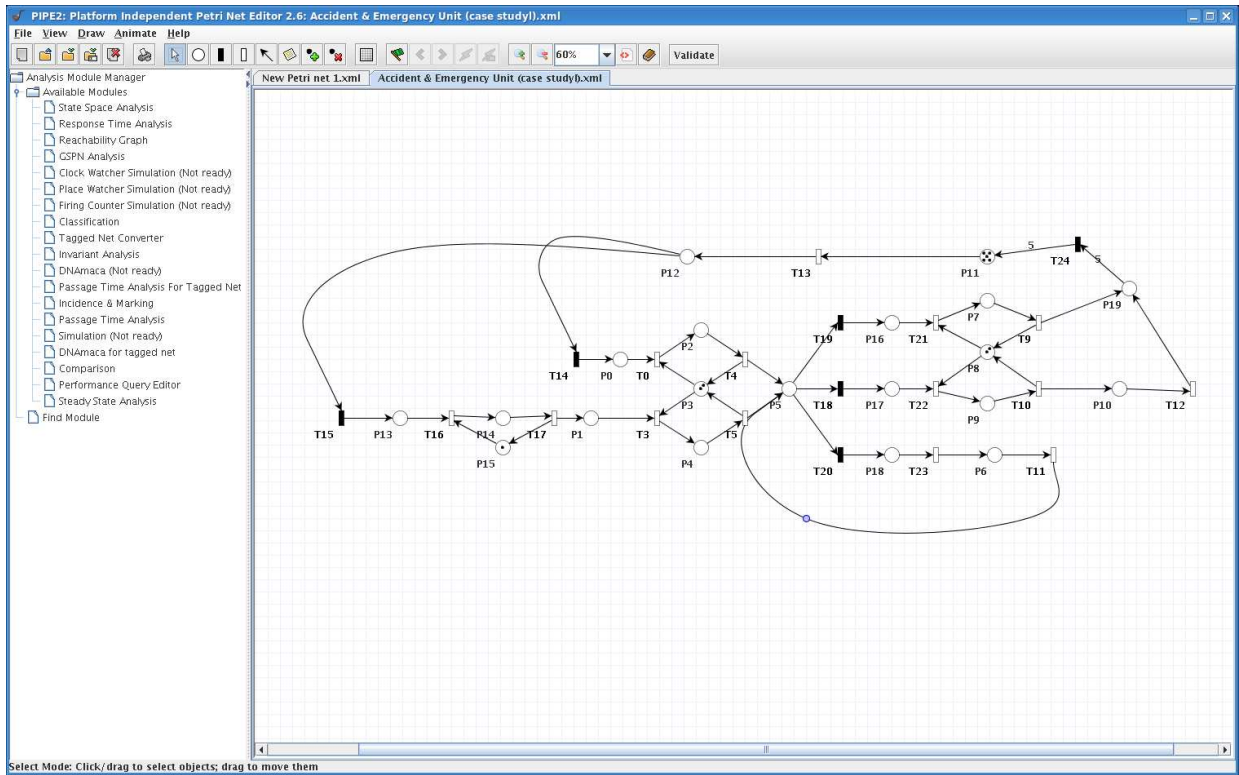


Figure 1: *PIPE2*'s GSPN design interface.

els. A panel to the left of the canvas enables users access to these modules. Users are encouraged, however, to implement their own modules to tailor the tool's functionality to their requirements. The modules discussed below are all included in the current *PIPE2* release.

Structural Analysis Modules

Model Classification Module: Classifies GSPN models based on their structure into the following categories: state machine, marked graph, free choice net, extended free choice net, simple net and extended simple net.

Model Comparison Module: Compares two GSPN models based on attributes determined by users as comparison criteria.

State Space Module: Determine properties of GSPN models such as liveness, boundedness and existence of deadlocks.

Incidence & Marking Module: Determines and displays the forward and backward incidence matrices and the initial marking.

Reachability Graph Module: Provides a visual representation of a GSPN model's underlying reachability graph.

Performance Analysis Modules

Simulation Module: Studies the performance of models by investigating the average number of tokens per place and mean transition throughputs, using Monte Carlo simulation.

Steady-State Analysis Module: Calculates state and count measures from the steady-state distribution via an interface to the *DNAmaca* [17] steady state analyser.

Passage Time Analysis Module: Calculates probability density and cumulative distribution functions for the time

taken for a model to complete a user-defined passage via an interface to the *SMARTA* [18] passage time analyser.

GSPN Analysis Module: Calculates analytically the distribution of tokens on places, and the mean throughput of timed transitions.

Additionally, recent work on specifying customer-centric performance queries using the concept of "tagged tokens" [19] is currently being integrated in *PIPE2*, in the form of modules for the steady-state and passage time analysis of GSPNs with tagged tokens.

3. PERFORMANCE TREES

Performance Trees are a formalism for the graphical specification of performance queries. A Performance Tree query is represented as a tree structure that consists of nodes and connecting arcs. Nodes can be of two kinds: *operation* nodes represent performance-related functions, such as the calculation of passage time densities, while *value* nodes are instances of basic types such as sets of states, actions, and numerical/boolean constants. Complex queries can be constructed by connecting nodes together. Figure 2 shows an example Performance Tree query that asks: "Is it true that the passage between the set of states 'start' and the set of states 'target' takes less than 5 time units with a probability of at least 0.98?"

Table 1 shows the currently available Performance Tree operation and value nodes. Performance Trees also support macros, which allow custom performance concepts to be defined by users using existing nodes.



























Textual	Graphical	Description
?		The result of a performance query.
Mult		Concurrent evaluation of multiple independent queries.
PTD		Passage time density, calculated from a given set of start and target states.
Dist		Passage time distribution obtained from a passage time density.
Perctl		Percentile of a passage time density or distribution.
Conv*		Convolution of two passage time densities.
ProbInInterval		Probability with which a passage takes place in a certain amount of time.
ProbInStates*		Transient probability of a system being in a given set of states at a given point in time.
Moment		Raw moment of a passage time density or distribution.
FR		Mean occurrence of an action (mean firing rate of a transition).
SS:P		Probability mass function yielding the steady-state probability of each possible value taken on by a StateFunc when evaluated over a given set of states.
SS:S*		Set of states that have a certain steady-state probability.
StatesAtTime*		Set of states that the system can occupy at a given time.
InInterval		Boolean operator that determines whether a numerical value is within an interval.
Macro*		User-defined performance concept composed of other operators.
\subseteq		Boolean operator that determines whether a set is included in or corresponds to another set.
\vee/\wedge		Boolean disjunction or conjunction of two logical expressions.
\neg		Boolean negation of a logical expression.
\bowtie		Arithmetic comparison of two numerical values.
\oplus		Arithmetic operation on two numerical values.
Num		A real number.
Range		A range of real numbers, defined by a lower and an upper bound.
Bool		A Boolean value.
Actions		A system action.
States		A set of system states.
StateFunc		A real-valued function on a set of states.

Table 1: Performance Tree nodes (*denotes feature currently under development in PIPE2)

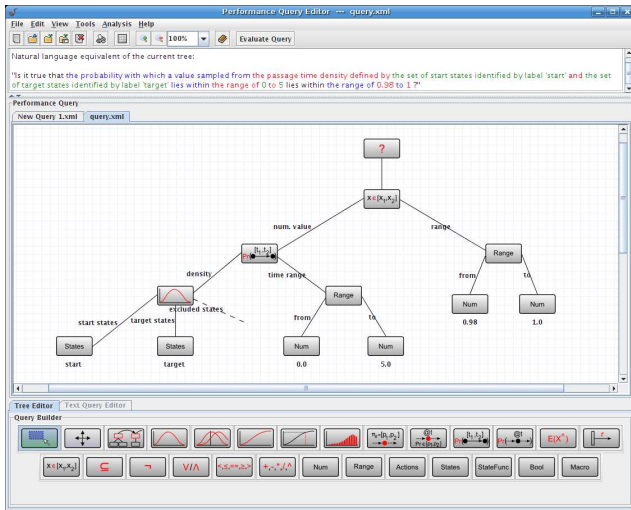


Figure 2: An example Performance Tree, shown in PIPE2's Performance Query Editor.

4. PARALLEL AND DISTRIBUTED PERFORMANCE ANALYSIS WITH PIPE2

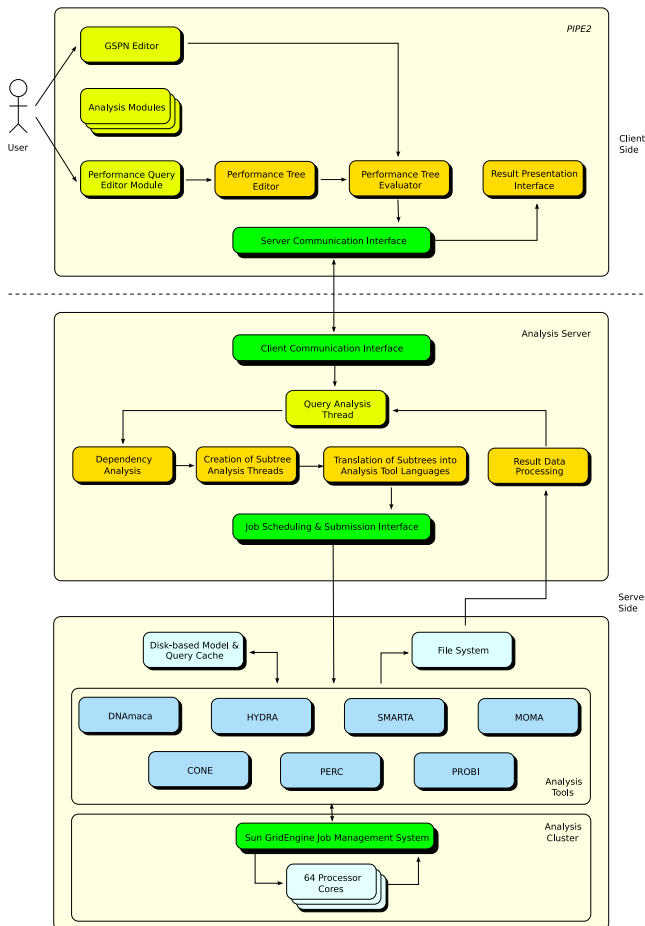


Figure 3: Performance Analysis Environment Architecture.

Figure 3 shows the integrated performance analysis environment that provides users with the ability to design system models and to specify and evaluate performance queries. The environment consists of four components: the Analysis Client (*PIPE2*), the Analysis Server, a set of Analysis Tools and the Analysis Cluster.

Users interact directly with the client side implemented in *PIPE2*, which allows them to create GSPN models and corresponding Performance Tree queries. The *PIPE2* client communicates with the Analysis Server, which in turn interacts with a range of analysis tools hosted on a computing cluster to calculate the answer to the user's query.

Analysis Client

PIPE2 has the role of the client within the analysis environment, and is its only user-facing component. It is the gateway to the functionality provided by the analysis environment's other components, and allows users to create system models and Performance Tree queries. When the user initiates query evaluation, it communicates model and query data to the Analysis Server for further processing. As soon as the Analysis Server returns evaluation results, *PIPE2* receives these and presents them visually to users.

Analysis Server

The Analysis Server is responsible for handling evaluation requests issued by *PIPE2*, and the coordination of the subsequent performance query evaluation process. It is deployed on the Analysis Cluster's primary host, and is continuously available to accept incoming analysis requests. The Analysis Server decomposes performance queries into subtrees and sends these to specialised Analysis Tools for evaluation.

Analysis Tools

The evaluation of quantitative measures defined in Performance Tree queries is ultimately carried out by a set of Analysis Tools that are invoked by the Analysis Server. Tools that form part of the analysis environment are:

DNAmaca [17] – a Markov chain steady-state analyser that can solve models with up to $O(10^8)$ states. It performs functional and steady-state analyses, and computes performance statistics, such as the mean, variance and standard deviation of expressions computed on system states. In addition, it also calculates mean rates of occurrence of actions. The raw distribution from which these performance statistics are calculated can also be obtained. *DNAmaca* is used for the evaluation of the SS:P and FR Performance Tree operators.

SMARTA [18] – a distributed MPI-based semi-Markov response time analyser that performs iterative numerical analyses of passage times in very large semi-Markov models (including GSPNs), using hypergraph partitioning and numerical Laplace transform inversion. *SMARTA* is suitable for the analysis of the PTD and Dist operators on GSPN models where start and target states are vanishing.

HYDRA [18] – a distributed Markovian passage time analyser that uses hypergraph partitioning and uniformisation techniques. *HYDRA* is suitable for the evaluation of the PTD and Dist Performance Tree operators, and also features transient analysis capabilities that are useful for the evaluation of the ProbInStates and StatesAtTime operators.

MOMA [20] – an n^{th} order passage-time raw moment calculator for GSPN models that uses a Laplace transform-

based method. *MOMA* is used for the evaluation of the Moment operator when applied to a passage time density.

CONE [20] – a performance analyser that, together with *SMARTA*, evaluates the convolution of two passage time densities using a Laplace transform-based approach. *CONE* is used for the evaluation of the Conv operator.

PERC [20] – a performance analyser that calculates percentiles of passage time distributions and densities. It works in conjunction with *SMARTA* and is used for the evaluation of the Perctl operator.

PROBI [20] – a performance analyser that calculates the probability with which a value sampled from a passage time density lies within a certain interval. *PROBI* is used for the evaluation of the ProbInInterval operator.

Analysis Cluster

Camelot, the computational cluster forming the backbone of the analysis environment, consists of 16 dual-processor dual-core nodes, each of which is a Sun Fire x4100 with two 64-bit Opteron 275 processors and 8GB of RAM. Nodes are connected with both Gigabit Ethernet and Infiniband interfaces. The Infiniband fabric runs at 2.5Gbit/s, and is managed by a Silverstorm 9024 switch. Job submission is handled by Sun GridEngine (SGE), a Grid management middleware that configures and exposes *Camelot* as a computational Grid resource. Clients submit sequential and parallel (MPI) jobs to SGE via the Distributed Resource Management Application API (DRMAA).

5. PERFORMANCE QUERY EVALUATION

Users interact with *PIPE2* to design system models and performance queries. The tool also enables them to initiate the automatic evaluation of performance queries through a single button click, and provides them with an evaluation progress tracking and visual result feedback mechanism. When a user requests a query’s evaluation, *PIPE2* establishes a connection with the Analysis Server in the background, which in turn delegates the processing of individual queries to dedicated analysis threads.

Analysis threads process serialised versions of system models and performance queries and construct an internal representation of the data. They decompose queries into a set of subtrees, and subsequently create helper threads for each subtree. Once a helper thread has been created, it submits its subtree for evaluation in the form of an analysis job to SGE. Analysis jobs consist of analysis tool invocation requests, based on the types of subtree nodes. Threads communicate with SGE via a DRMAA interface. SGE has built-in scheduling algorithms that are used to distribute jobs onto available processors on the analysis cluster.

In the case where the evaluation of a subtree is conditional on results to be obtained from the evaluation of other subtrees, the job is suspended until all required inputs are available. Note that parallelism takes place on two levels during evaluation. Firstly, certain tools are able to carry out the evaluation of individual computation-intensive Performance Tree nodes in a parallelised manner. Secondly, if nodes within the query tree are independent of each other, they can be evaluated concurrently.

To avoid redundant work, the Analysis Server incorporates a disk-based caching mechanism that stores performance query evaluation results. In order to differentiate between multiple queries on the same model, MD5 hashes of

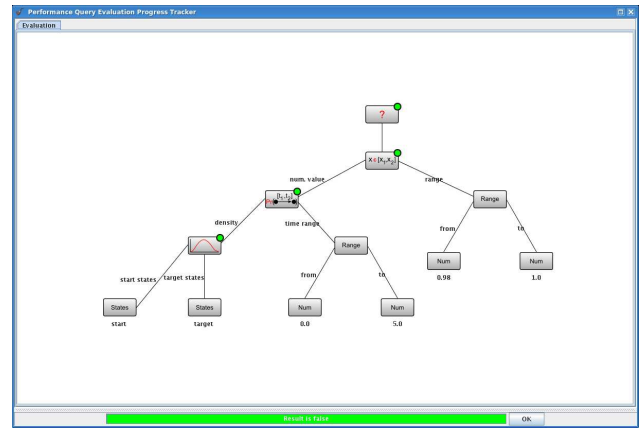


Figure 4: The result of evaluating the Performance Tree in Figure 2 using *PIPE2*.

the model description and the performance query specification are calculated for each query. These are used to create a two-level structure in which the computed performance measures can be stored. Before any computation takes place, a cache look-up for the hash of the given model is performed. If a match is found, the hash of the current query is compared to all hashes of queries in the cache that have been evaluated on that particular model. A match indicates that the query results can be retrieved from the cache. No match means that the query needs to be evaluated.

Users can configure the analysis with regards to the number of processors that are to be used during evaluation and also whether or not caching should be enabled. Users can also specify the time ranges of interest for measures such as passage time densities, or request that these are automatically computed to show the main region of probability mass. For more details of these mechanisms, see [7, 20].

As soon as each subtree evaluation completes, the Analysis Server forwards its result to *PIPE2* to be displayed to the user. This enables the user to inspect partial results before overall evaluation completes.

6. CASE STUDY

Figure 1 shows a GSPN model of a hospital’s Accident and Emergency (A&E) department after it has been input into *PIPE2*. The model describes a system with the following behaviour. An initial number of healthy individuals (on P_{11}) fall ill at a certain rate and either go to the hospital themselves (via T_{14}), in which case they are categorised as walk-in patients, or place an emergency call to request an ambulance (via T_{15}). Once they have reached A&E, walk-in patients wait until they can be seen by a nurse for initial assessment, while ambulance patients are loaded onto a trolley on which they wait until a nurse becomes available. Nurses assess ambulance patients with priority. After initial assessment, patients (now on P_5) proceed to either be seen by a doctor (via T_{19}), be taken for emergency surgery (via T_{18}), or be sent for laboratory tests (via T_{20}). Once a patient is discharged (via T_9 or T_{12}), they are assumed to be healthy again.

The model is parameterised with P , N , D and A , which denote the number of patients (on P_{11}), nurses (on P_3), doctors (on P_8) and ambulances (on P_{15}), respectively. Here,

we set $P = 5$, $N = 2$, $D = 2$ and $A = 1$. We wish to know whether or not the hospital is capable of processing all patients within 5 hours with 98% certainty, and therefore use a Performance Tree query of the form shown in Figure 2. We define our start states as those where all five patients are on place P_{11} and our target states as those where all five patients are on place P_{19} .

Figure 4 shows *PIPE2*'s evaluator window when this query has successfully completed analysis (shown by traffic light indicators against all nodes that require computation). The overall result ("false") can be found by clicking on the top-most ("?") node. The user can also click on other nodes to view sub-results – for example, clicking on the PTD node will display the full probability density function of the computed passage time.

7. CONCLUSIONS AND FUTURE WORK

In this paper we have presented an overview of *PIPE2*, a GSPN-based modelling and performance analysis tool. We have described its functionality, including its model design facilities, and have introduced the distributed analysis environment that provides extensive performance evaluation features. We have described the components of this analysis environment, and have demonstrated how *PIPE2* has been extended to support the graphical creation of Performance Tree queries, and how these queries are evaluated on a parallel and distributed analysis cluster. Future development will include the integration of a *PEPA*-based stochastic process algebra model specification module and natural language-based performance query specification. We will also be developing methods for the optimisation and efficient scheduling of computations in order to achieve improved analysis cluster response times.

8. REFERENCES

- [1] PIPE2: Platform-Independent Petri net Editor – <http://pipe2.sourceforge.net>.
- [2] M. Ajmone Marsan, G. Conte, and G. Balbo, "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems," *ACM Transactions on Computer Systems*, vol. 2, no. 2, pp. 93–122, May 1984.
- [3] P. Bonet, C. Lladó, R. Puigjaner, and W. Knottenbelt, "PIPE v2.5: A Petri net tool for performance modelling," in *Proc. 23rd Latin American Conference on Informatics (CLEI'07)*, San Jose, Costa Rica, October 2007.
- [4] M. Melià, C. Lladó, R. Puigjaner, and C. Smith, "An experimental framework for PIPE2," in *Proc. 5th International Conference on the Quantitative Evaluation of Systems (QEST'08)*, St. Malo, France, 2008, pp. 239–240.
- [5] T. Suto, J. T. Bradley, and W. J. Knottenbelt, "Performance trees: A new approach to quantitative performance specification," in *Proc. 14th Intl. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'06)*, Monterey, CA, September 2006, pp. 303–313.
- [6] —, "Performance trees: Expressiveness and quantitative semantics," in *Proc. 4th Intl. Conf. on the Quantitative Evaluation of Systems (QEST'07)*, Edinburgh, UK, September 2007, pp. 41–50.
- [7] D. K. Brien, N. J. Dingle, W. J. Knottenbelt, H. Kulatunga, and T. Suto, "Performance trees: Implementation and distributed evaluation," in *Proc. 7th Intl. Workshop on Parallel and Distributed Methods in Verification (PDMC'08)*, Budapest, Hungary, March 2008, Work in Progress Report.
- [8] J. Guillen-Scholten, F. Arbab, F. de Boer, and M. Bonsangue, "Modeling the exogenous coordination of mobile channel-based systems with Petri nets," *Electronic Notes in Theoretical Computer Science*, vol. 154, no. 1, pp. 121–138, 2006, Proc. 4th International Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA'05).
- [9] F. Bernardini, M. Gheorghe, F. Romero-Campero, and N. Walkinshaw, "A hybrid approach to modelling biological systems," in *Proc. 8th Workshop on Membrane Computing*, vol. LNCS 4860. Springer, 2007, pp. 138–159.
- [10] R. Chitrakar, "BPEL (Business Process Execution Language) – Specification, modelling and analysis," Master's thesis, Univeristy of Illinois at Chicago, 2006.
- [11] A. Peñarroya, F. Casado, and J. Rosell, "A performance analysis tool of discrete-event systems," Institute of Industrial and Control Engineering, Technical University of Catalonia, Tech. Rep. IOC-DT-P-2007-1, 2007.
- [12] O. Bonnet-Torrés, P. Domenech, C. Lesire, and C. Tessier, "Exhost-PIPE: PIPE extended for two classes of monitoring Petri nets," in *Proc. 27th International Conference on Application and Theory of Petri Nets (ICATPN'06)*, vol. LNCS 4024. Springer, 2006, pp. 391–400.
- [13] The Petri Net Markup Language – <http://www2.informatik.hu-berlin.de/top/pnml/>.
- [14] D. Gašević, V. Devedžić, and N. Veselinović, "P3 – Petri net educational software tool for hardware teaching," in *Proc. 10th Workshop Algorithms and Tools for Petri Nets*, Eichstätt, Germany, 2003, pp. 111–120.
- [15] WoPeD: Workflow Petri Net Designer – <http://www.woped.org/>.
- [16] H. Verbeek and W. van der Aalst, "Woflan 2.0: A Petri-net-based workflow diagnosis tool," in *Proc. 21st International Conference on Application and Theory of Petri Nets (ICATPN'00)*, vol. LNCS 1825. Springer, 2000, pp. 475–484.
- [17] W. J. Knottenbelt, "Generalised Markovian analysis of timed transitions systems," M.Sc. Thesis, University of Cape Town, South Africa, July 1996.
- [18] N. Dingle, "Parallel computation of response time densities and quantiles in large Markov and semi-Markov models," Ph.D. dissertation, Imperial College London, United Kingdom, 2004.
- [19] N. Dingle and W. Knottenbelt, "Automated customer-centric performance analysis of Generalised Stochastic Petri nets using tagged tokens," in *Proc. Practical Application of Stochastic Models (PASM'08)*, Majorca, Spain, September 2008.
- [20] D. K. Brien, "Performance trees: Implementation and distributed evaluation," M.Sc. Thesis, Imperial College London, United Kingdom, June 2008.