Proceedings

»PNSE'17«

International Workshop on

Petri Nets and Software Engineering

Satellite event of the

38th International Conference on Application and Theory of Petri Nets and Concurrency

---

17th International Conference on Application of Concurrency to System Design

Zaragoza, Spain, June, 2017

Including the proceedings of the co-located event

»MoSEBIn'17«

International Workshop on

Modeling and Software Engineering in Business and Industry

Editors: Daniel Moldt,
Lawrence Cabac and
Heiko Rölke

Proceedings of the International Workshop on

P etri
N ets and
S oftware
E ngineering
PNSE'17

Editors: Daniel Moldt,
Dirk Fahland,
Andreas Solti and
Laura García-Borgoñón

Proceedings of the International Workshop on

Mo deling and
S oftware
E ngineering in
B usiness and
In dustry
MoSEBIn'17

University of Hamburg
Department of Informatics

Compilation Editor:
Daniel Moldt
University of Hamburg
Department of Informatics
Vogt-Kölln-Str. 30
D-22527 Hamburg
Germany
http://www.pnse.de
http://www.Petri-Net.de

These proceedings are published online by the editors as Volume 1846 at

CEUR Workshop Proceedings
ISSN 1613-0073
http://ceur-ws.org/Vol-1846/

Prefaces

These are the proceedings of the International Workshop on *Petri Nets and Software Engineering* (PNSE'17) in Zaragoza, Spain, June 25–26, 2017 which also includes the papers of the International Workshop on *Modeling and Software Engineering in Business and Industry* (MoSEBIn'17). Both workshops are co-located events of

- *Petri Nets 2017* – the 38th International Conference on Applications and Theory of Petri Nets and Concurrency and
- *ACSD 2017* – the 17th International Conference on Application of Concurrency to System Design.

More information about the workshop can be found at

`http://www.informatik.uni-hamburg.de/TGI/events/pnse17/`
and
`http://www.informatik.uni-hamburg.de/TGI/events/mosebin17/`

PNSE'17 preface:

For the successful realization of complex systems of interacting and reactive software and hardware components the use of a precise language at different stages of the development process is of crucial importance. Petri nets are becoming increasingly popular in this area, as they provide a uniform language supporting the tasks of modeling, validation and verification. Their popularity is due to the fact that Petri nets capture fundamental aspects of causality, concurrency, synchronization and choice in a natural and mathematically precise way without compromising readability. The use of Petri nets (P/T-nets, Coloured Petri nets and extensions) in the formal process of software engineering, covering modeling, validation, execution, simulation and verification, is presented as well as their application in several domains and tools supporting the disciplines mentioned above.

MoSEBIn'17 preface:

The workshop is a forum for those interested in modeling, especially of, for and within business and industry environments. Business and industry environments are important and relevant application domains for modeling and software engineering. Both academics and practitioners can contribute and learn from such a meeting. The fundamental interest is to understand modeling within this area and what environments and applications actually demand from modelers and software engineers.

Communication between users and software engineers is based on models, therefore, the transformation from application domain models to computer science is a major task that we want to discuss during the workshop from many perspectives. Furthermore, software engineering for business and industry has to provide solutions that have to fit special needs of the people in these fields.

The mutual dependencies, services, requirements, expectations, solutions etc. between software engineers and business people / people from industry shall be discussed during the workshop.

Last but not least in the context of any organisational institution the roles of modeling within software engineering and how to use software engineering for modeling can also be addressed from various perspectives.

For both workshops we have chosen José Ángel Bañares and Julia Padberg as invited speakers. We received eighteen high-quality contributions for these proceedings. The program now consists of ten full papers, two short papers, three poster contributions and two invited talks.

The international program committee of PNSE'17 was supported by the valued work of Alfredo Capozucca, Stefan Klikovits, Artur Niewiadomski, Marcin Piątkowski and Benoît Ries as additional reviewers. Their valuable work is highly appreciated.

Furthermore, we would like to thank our colleagues in the local organization team at the Aragón Institute of Engineering Research (I3A), Zaragoza University, Spain for their support.

The organzational/technical work in Hamburg was supported by Louis Kobras, Michael Haustermann and Dennis Schmitz.

Without the enormous efforts of authors, reviewers, PC members and the organizational teams, this workshop would not provide such an interesting booklet.

Thank you very much!

Danial Moldt, Lawrence Cabac, Heiko Rölke
Hamburg, June 2017

PNSE'17 Program Committee

Elvio Gilberto Amparore	(Italy)
Kamel Barkaoui	(France)
Robin Bergenthum	(Germany)
Didier Buchs	(Switzerland)
Lawrence Cabac (Co-Chair)	(Germany)
Piotr Chrzastowski-Wachtel	(Poland)
Gianfranco Ciardo	(USA)
José-Manuel Colom	(Spain)
Jörg Desel	(Germany)
Raymond Devillers	(Belgium)
Giuliana Franceschinis	(Italy)
Nicolas Guelfi	(Luxembourg)
Stefan Haar	(France)
Serge Haddad	(France)
Xudong He	(USA)
Kunihiko Hiraishi	(Japan)
Gabriel Juhás	(Slovakia)
Peter Kemper	(USA)
Astrid Kiehn	(India)
Hanna Klaudel	(France)
Radek Koci	(Czech republic)
Maciej Koutny	(United Kingdom)
Lars Kristensen	(Norway)
Michael Köhler-Bußmeier	(Gemany)
Robert Lorenz	(Germany)
Łukasz Mikulski	(Poland)
Daniel Moldt (Co-Chair)	(Germany)
Berndt Müller	(Great Britain)
Wojciech Penczek	(Poland)
Laure Petrucci	(France)
Lucia Pomello	(Italy)
Heiko Rölke (Co-Chair)	(Germany)
P.S. Thiagarajan	(USA)
Yann Thierry-Mieg	(France)
Henricus M.W. (Eric) Verbeek	(Netherlands)
Karsten Wolf	(Germany)

MoSEBIn'17 Program Committee

María Alpuente	(Spain)
Ahmed Awad	(Egypt)
Bernhard Bauer	(Germany)
Olivier Boissier	(France)
Christine Choppy	(France)
Claudio Di Ciccio	(Austria)
M.J. Escalona	(Spain)
Dirk Fahland (Co-Chair)	(Netherlands)
Walid Fdhila	(Austria)
Michael Felderer	(Austria)
Luciano García-Bañuelos	(Estonia)
Laura García-Borgoñón (Co-Chair)	(Seville)
Holger Giese	(Germany)
José González Enríquez	(Spain)
Vincent Hilaire	(France)
Lom Messan Hillah	(France)
Benjamin Hirsch	(United Arab Emirates)
Martin Kollingbaum	(United Kingdom)
Gabriele Kotsis	(Austria)
Maristella Matera	(Italy)
Massimo Mecella	(Italy)
Daniel Moldt (Co-Chair)	(Germany)
Andreas Oberweis	(Germany)
Andrea Omicini	(Italy)
Pascal Poizat	(France)
Mattia Salnitri	(Italy)
Stefan Schönig	(Germany)
Arik Senderovich	(Israel)
Andreas Solti (Co-Chair)	(Austria)
Ulrich Ultes-Nitsche	(Switzerland)
Antonio Vallecillo	(Spain)
Jan Martijn Van Der Werf	(Netherlands)
Mathias Weske	(Germany)
Manuel Wimmer	(Austria)
Christian Zirpins	(Germany)

Contents

Part I

Invited Talks

Model-Driven Development of Performance Sensitive Cloud Native Streaming Applications

José Ángel Bañares

Computer Science and Systems Engineering Department
COS2MOS research group, Aragón Institute of Engineering Research (I3A)
University of Zaragoza, Zaragoza, Spain
banare@unizar.es

Abstract. The number of applications that process data in a stream basis has increased significantly over recent years due to the proliferation of sensors. Additionally, in cyber-physical systems, physical and software components are deeply intertwined, adding the ability to act on the environment.
In many cases, cloud resources are used for the processing, exploiting their flexibility, but these sensor streaming applications often need to support operational and control actions that have real-time and low-latency requirements that go beyond the cost effective and flexible solutions supported by cloud platforms. The development of these applications cannot be delegated to the magical properties of frameworks and services that promise simple solutions, hiding the inherent underlying complexity of cloud resources. It raises the difficulty of developing complex streaming processing in the cloud and highlights the need for a suitable developing methodology. Moreover, during the developing lifecycle, a number of facets have to be considered such as the design of functional parallel solutions, the impact of a target cloud platform that exhibits different degrees of performance variability, or the need for more complex performance requirement support. This talk will present our experiences in developing Petri Net models for performance sensitive cloud applications thus leveraging the use of formal models in complex scenarios.

Keywords: Streaming applications · Cloud Native Applications · Performance Sensitive Applications · Petri nets.

1 Cloud Native Streaming Applications

There is an emerging interest in low latency streaming applications that consume big volumes of data. Stream processing finds application in almost every industry, business and scientific application. The sources of data can be generated from sensors, scientific instruments, simulations, social networks, business processes, etc. Data are transmitted continuously, forming a sequence of data elements known as a data stream and must be processed fast in order to control systems, take corrective/strategic actions, or react to urgent situations.

To effectively support this emerging class of applications, it is often necessary to generate workflow specifications that can be dynamically adapted – as new data becomes available in the context of the mainstream definition of Big Data as the three Vs: volume, variety and velocity [1]. With the elastic nature of many cloud environments enabling such dynamic workflow graphs to be enacted more efficiently, it might seem that data flow applications would somehow become simpler, but that is not the case. One of the challenges of data flow applications is that they must be designed with the needed level of dynamism to take account of the availability of data and the variability of the execution environment, which can be dynamically scaled out based on demand. According to [2], clouds may also be used as accelerators to improve the application time-to-completion, or to handle unexpected situations such as an unanticipated resource downtime, inadequate allocations or unanticipated queuing delays. To achieve such system, developers must meet several challenges that go beyond pure functionality. A trial-and-error implementation that tries different deployments of a streaming application over different distributed computing platforms is far from a trustworthy solution.

A cloud native application (CNA) is designed specifically to take full advantage from the cloud computing characteristics. Fehling et al. have identified the main characteristics of a cloud computing architecture in [3]: 1) The **decomposition** of the functionality in chunks of distributed functionality in such a way that that each component that made up the application can be scaled out independently. 2) The design space or operation model comprises the analysis of the application **workloads**, and the identification of how the application handles **state** and the cost of sharing information. 3) Applications are desgined taking into account **resilience** and **elasticity**: An important aspect related to the workload is to consider the dynamic nature of the cloud, which can be caused by performance variation of machines, services competiting for shared resources, and changing user quality of services requirements [4, 5].

The need for more complex performance sensitive applications callenges current development practices. The complexity of developing cost-effective and performance sensitive cloud native streaming applications has been addressed from different apporaches: 1) Cloud based frameworks that lift the level of abstraction reducing complexity and hiding resource management. 2) Collecting high-quality solutions, which are presented as patterns, to recurring problems in parallel applications and cloud platforms. 3) Developing ad-hoc performance models to predict the behavior of particular patterns on specific platforms. As far as this author knows, there is a lack of complete methodological approaches to conduct developers through the entire process of developing streaming applications as CNA, beyond the partial solutions provided by these approaches. The main shortcoming of current approaches is the non-use of formal models for helping developers to reasoning and automatize the proces of analyisis of functional and non-functional requirements. This paper covers the main characteristics of a cloud native streaming applications making reference to our works using Petri Net models for performance sensitive cloud applications.

First the requirements and a synoptical view of the proposed methodology is presented in section 2. Section 3 briefly presents a specification language to support the methodology. The behaviour of component specifications is defined by Petri Nets to support the analysis and simulation of models. Due to the large size and complexity of these systems, it is not possible to develop detailed models. We will illustrate how to use models for understanding complex behaviors, and how to calculate performance boundaries and conduct simulations in section 4. Finally, in section 5, PN models of different mechanisms to support *elasticity* and *resilience* are presented. This models allow developer to validate the mechanisms in different simulated scenarios.

2 Methodology for Performance Sensitive Streaming Applications on the Cloud

In [6–9] we have identified and presented the principles of the methodology to cope with the inherent complexity of streaming applications on the cloud. The methodology considers all involved elements at different abstraction levels.

We have identified the following *modelling requirements* that go beyond pure functionality:

- A **development process**, which is guided by the identified abstraction levels, that provides a number of modelling artifacts, analytical methods, and guidelines to support it. The methodology must address functional and non-functional requirements together with the specification of the execution infrastructure and the involved resources.
- A **specification language** to describe a streaming application as a collection of platform-independent building blocks. The language must support complementary point of views: behavioral specification of concurrent processes, transformations operated over the data flow, and structural description of components that configure the application [10–14].
- A **formal component-based** development to build models from existing components and capability to reason about the resulting composition. Reuse of components allows developers to use knowledge of their properties to predict the new system properties. Components models must provide a rich specification to facilitate the use of different analysis and prediction techniques that simplify a system design while increasing trust in its correct implementation [15].
- A **guide** of the possibilities of model reasoning for efficient and reliable design and / or optimization, combining simulation, and approximate analysis. The specification must be executable to support both *analytical analysis* and *simulation* in a synergic way.

A description of the global functionality by means of an algorithm says nothing about the structure or the components that make up the system. There are many ways in which a system can be built to provide the same functionality with different concurrent behaviours and different deployments over distributed

infrastructures [11]. Our approach to identify the elements to compound our hybrid specification can be summarised with the equation 'Specification of CDFA = Functional Entities + Communication/Synchronisation mechanisms + Data Dependencies + Resources'. The identification and characterisation of each building block of the proposed equation constitutes the basic specification element.

A *synoptical view* of our methodology include the following steps:

- **Functional Level.** The process starts by identifying the functional requirements of the problem domain and the outcome of this step is a functional model.
- **Qualitative Analysis.**The functional model analysis is used to gain identify problems and help guide the redesign of the functional model aiming to achieve the maximum level of concurrency.
- **Operational Level.** The operational level takes into account the execution platforms, and it explores the design space to select the design pattern that most effectively defines how to map processes to resources. The outcome is an operational model.
- **Quantitative Analysis.** The integration of the functional and the operational model allows the designer to evaluate performance and reward functions. The analysis can help guide the redesign of the functional and operational models to meet non-functional requirements.
- **Implementation Level.** This stage transforms the model into a flat model of processes that are deployed in a topology of cloud resources.
- **Monitoring.** The last step collects monitoring data from all used resources and applications. Collected data and developed models can help identify performance anomalies, and provide support to the autonomic principles of a Platform as a Service. The primary aim is to reduce human intervention, cost, and the perceived complexity by enabling the autonomic platform to self-manage applications [16].

The semantics of the component-based language is defined formally in terms of ordinary PNs [17] in order to translate to the methodology all the advantages derived from a mathematically based model –e.g. Analysis, Verification, or Equivalence Relations. The consideration of PNs is based on the natural descriptive power of concurrency, but also on the availability of analytic tools coming from the domain of Mathematical Programming and Graph Theory. Moreover, taking into account that PNs are executable specifications, PN models can also be simulated.

3 Data flow language specification

In [9], we presented a component based specification LANGuage of Layers and tIERS (*Langliers*) to support the methodology for building trustworthy continuous data flow applications. *Langliers* allows developers to specify the functional model as layers, the logical groupings of the functionality and components; and the operational model as tiers, the physical distribution of the functionality

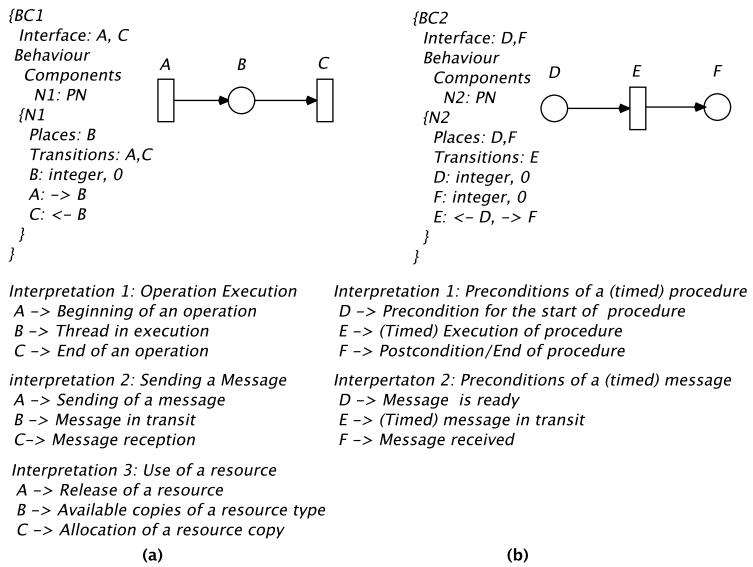


Fig. 1. Component specification of basic component primitives and interpretations.

and components on separated servers, computers, networks or remote locations. Component behaviour is defined by Petri Nets.

The constructive elements of a data stream application begins with the definition of the most basic building blocks and their interpretation as constructive primitives of distributed applications, and continues on their composition by means of simple operators, which provides the way to configure components with complex behaviours. *Langliers* represents the basic components identified in [7], **Computational** and **Data Transmission Processes**, to describe an event processing network as a graph showing various connected processing components that operate over data stream. This processing network model is an abstraction that describes the **functional** behaviour of an streaming application made up of a number of platform-independent components. The explicit network specification allows developers to visualize the functional model and apply analysis techniques. The graphical representation of a large or complex network can be somewhat unwieldy, but an hierarchical approach where components can be defined by the composition of subcomponents can simplify the specification of large models. The last remaining step to specify a complete model is to set the implementation of the functional model with the specification of the resources that will be used to execute the model. This **operational** specification can be used to conduct performance optimizations selecting a good mapping of Computational Processes and Data Transmission Processes to computational and network resources.

In *Langliers* a component is expressed in the form of an *interface* and a *behaviour* description. The **interface** specifies the services the component pro-

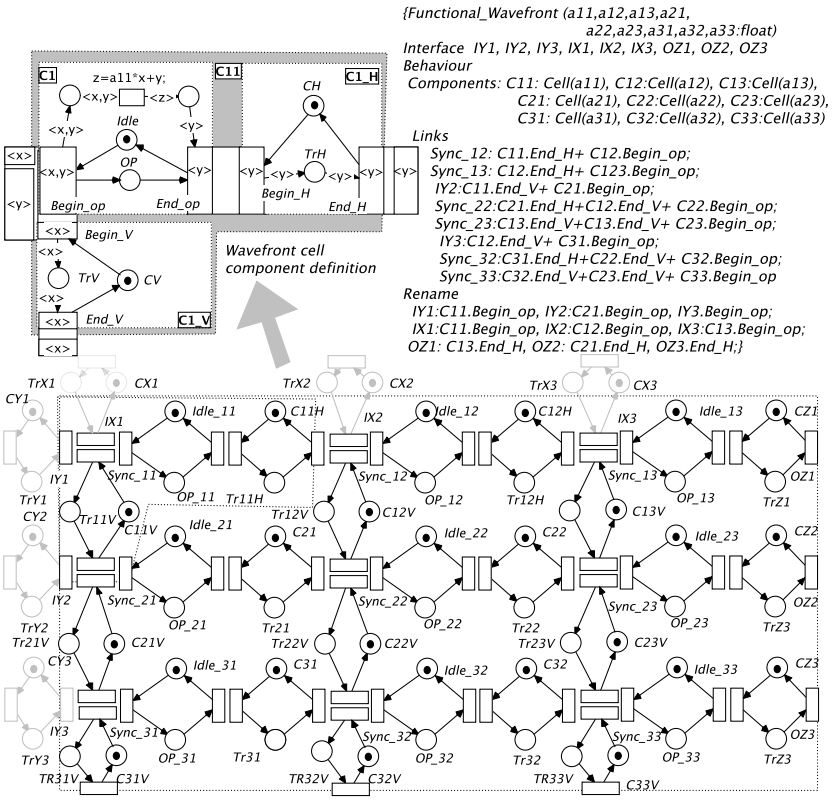


Fig. 2. Functional 3×3 wavefront array.

vides, and publishes its input and output **ports**. It gives information at the syntax level that enables *data type checking*, and publishes the *events* that trigger computations and state changes. This way, our data stream model follows a data driven execution model where events lead to computations that may generate events on other components. The ***behaviour*** represents components internal states and state changes. A component behavior description is specified either by one explicit PN or by the PN resulting from the behavior composition of subcomponents. It is inspired in digital systems VHDL language specification whose components specifications consist of a port declaration in the interface enumerating the events that change the component state, and the *architecture* declaration that describes either the entity's behavior or its structure. Samples of the proposed language are shown in figure 1 that shows the definition of basic component primitives and their interpretations, and figure 2 that shows the functional model of a 3x3 Wavefront array which is built with simple computational and data transmission processes.

4 Modeling for Understanding versus Modeling for Forecasting

The proposed model-driven methodology aims at providing different analysis and prediction techniques that allow developers to assess functional and non-functional properties by means of **qualitative** and **quantitative** analysis. Qualitative analysis aims to detect qualitative properties of concurrent and distributed systems, that is, to decide whether the model is correct and meets the given qualitative functional properties (e.g deadlock freedom). Qualitative PN analysis can be conducted by means of different techniques: (i) The construction of the state space of the model (reachability analysis) providing a complete knowledge of all its properties – in case state explosion does not hamper the use of this technique; (ii) Structural techniques in order to reason about some properties of the model, from the structure of the net.

In order to illustrate our methodology, we are making use of the Matrix-Vector Multiplication problem in streaming fashion, in particular, the *Wavefront Algorithm*, which represents a simple solution for large arrays [12]. The adequacy of the wavefront use case is that it is easy to formulate and present, but it shows a very complex behavior. Moreover, the literature shows a continuous effort of the research community to develop different performance models for the warfront algorithm executed on different platforms. The wavefront model is a strongly connected marked graph (a subclass of Petri nets in which each place has only one input and one output transition, being strongly connected in the sense of graph theory) [18]. The use of qualitative analysis can help to understand the behavior of the wavefront algorithm, and design solutions to obtain the maximum level of concurrency.

For quantitative analysis is importan to consider the dynamic nature of the cloud, which can be caused by performance variation of machine instances offering the same capability, and by services that are deployed, updated and destroyed all the time giving rise to a dynamic competition for shared resources [4, 5]. Due to the large size and complexity of these systems, it is not possible to develop a detailed model that captures all involved aspects. The usual approach to afford complexity in formal models such as queuing systems, is to model the essential aspects related with the behavior to be analysed, and to incorporate to the model the prediction of observable quantities by probabilistic models [19]. However, formal-model based analysis tools are only useful under certain assumptions that are not satisfied by cloud or container centres. Kazhei et al. point out three reasons that hamper the use of the tools of queuing theory: 1) The system size that involves a large number of nodes comparing to the number of nodes considered in traditional queuing analysis. 2) Involved service times must be modeled by a general probability distribution instead of a exponential distribution, which is a more convenient mathematical model. 3) The dynamic nature of these environments with dynamic loadworks and heterogeneous resources [20]. In these cases, we can use approximate methods to compute performance bounds, and complement these approximate analytical tools with simulations [20].

Streaming applications on the cloud are complex systems where customers and resources have not identical characteristics , and exponential distribution does not adequately model observed inter-arrival and service times. Therefore, traditional queuing systems are not feasible as forecasting models to obtain accurate performance evaluations. In a first approach, we labelled our model with all times following an exponential distribution. The net becomes a Stochastic Petri Net (SPN) and we translated it to the GreatSPN2.0.2 tool. The result obtained by the **GSPN analysis** showed a poor throughput. We can find the explanation of these poor results to the concurrency limitations found in the structural analysis. All computational resources will have the same throughput due to the high coupling imposed by synchronization constratins, and as a result throughput is determined by the slower resource. Intuitively, the use of a negative exponential probability distribution function which has a high coeffcient of variation with value 1, makes more likely a slower resource than the expected media with a larger number of resources.

In [18], authors present **upper** and **lower bounds** on the steady-state performance of marked graphs that can be computed effciently. In addition to the mean service time and mean inter-arrival time, the **coefficient of variation (CoV)** of resources and inter-arrival time has been proposed to introduce the dynamic nature of cloud applications and streaming applications on the cloud [4, 21, 20, 14]. Once these performance boundaries are defined, we can conduct **simulations** to explore peformance in function of the CoV of injection and processing rates. Hamzeh Khazaei et al. [20] proposes a CoV of delivery service on cloud centers with values between 0.5 and 1.4. According to these authors these values give reasonable insight into the behavior and dimensioning of cloud centers. **Simulations** showed that a CoV ranging from 0 to 1.5 covers all space of performance values between the calculated performance boundaries. With deterministic times we have performance near the upper bound, and with CoV of 1.5 performance is near the lower bound.

Previous analysis is adequated for analyse a concrete application or parallel pattern. However, to develop a complete model of each application that can be executed to obtain an operational and performance model of a cluster would be impractical. Profiling data is essential to feed models with time distribution annotations to forecast performance. For this purpose, it is necessary to provide a characterisation of different kind of applications and their effects on the remaining applications executed over the same resources. **Application profiling** is strongly associated with the workload analysis. Profiling must collect a large amount of data generated by the cloud resources and forecasting models are fed with these data to analyse resource contention and service degradation. A survey on forecasting and profiling models for cloud applications can be found in [22]. In[23, 24] we analyse performance of the Kubernetes system and develop a Reference net-based model of resource management within this **container management system**. Our model is characterised using real data from a Kubernetes deployment, and can be used as a basis to design scaleable applications that make use of Kubernetes.

5 Elasticity and Resilience

Finally, the two last properties to be considered in the development of CNA are **Component refinement** and **Management components** to support *elasticity* and *resilience*. Solutions for the latter are presented by patterns such as load balancers, or elastic queues. In [25, 26, 16, 27, 23] are presented specifications of strategies on cloud for resource management following autonomic principles at the application level for streaming and scientific workflows.

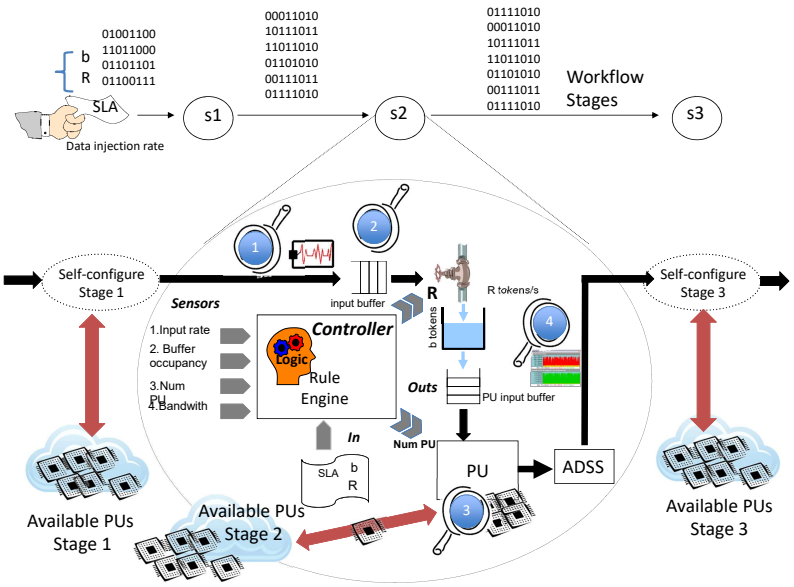


Fig. 3. System architecture and control loop for decision making in a processing node.

Formal models allow the rapid prototyping and simulation of different scenarios of complex mechanisms to support elasticy and resisilience. In [28, 29] we proposed a streaming workflow model of computation and an exception-handling mechanism for modifying at run-time the structure of a workflow. The need to close the control loop to take decisions and act on time requires a new model for analyzing and acting on IoT data that combines the cloud processing with *edge computing* or *Fog computing* [30], and **autonomic computing** techniques. It supposes the analysis of the most time-sensitive close to where data is generated and send selected data to the cloud. Additionaly, data elements are streamed from their source to their sink, and may be processed en-route (referred to **in transit** processing) [26]. This integration may imply that the associated runtime resource allocation is dependent on environmental conditions and can change for different enactments of the same workflow. In our proposal, our workflow specifications are independent of the constraints imposed by the resource allocation.

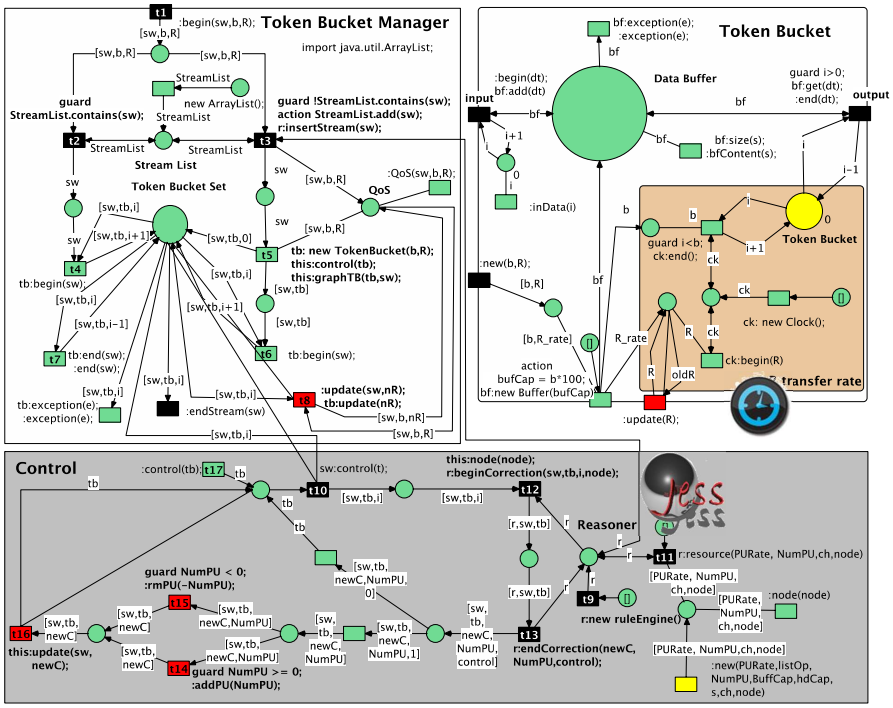


Fig. 4. Reference net model of the System architecture and control loop.

Finally, in [31] we proposed a profit-based resource management strategy for bursty data streams on shared Clouds. Even dynamic provisioning of resources may not be useful since the delay incurred might be too high – it may take several seconds to add new resources (e.g. instantiate new Virtual Machines (VMs)), and a scaling-up action might generate substantial penalties and overheads. We presented in this work an architecture and mechanisms based on the **token bucket** for the management of shared computational resources, in order to support QoS levels of several concurrent data streams and to maximize revenue of cloud providers. Figure 3 shows the system architecture and control loop for decision making in each processing node of a chain of distributed resources, and figure 4 the executable Reference net model specification implemented in Renew. Different scenarios were validated in terms of simulation. The fact that our Reference net models are executable, as they can be interpreted by Renew, allows us to use the same model to interface directly with OpenNebula from the nets: create and switch on and off real Virtual Machines (VMs), transmit data to the data centre and collect back the results. Our main contributions consists of data admission and control policies to regulate data access and manage the impact of data bursts, and a policy for resource redistribution that tries to minimize the cost of QoS penalty violation, maximising the overall profit.

Acknowledgments. This work was co-financed by the Industry and Innovation department of the Aragonese Government and European Social Funds (COS2MOS research group, ref. T93); and by the Spanish Ministry of Economy under the program "Programa de I+D+i Estatal de Investigación, Desarrollo e innovación Orientada a los Retos de la Sociedad", project id TIN2013-40809-R. I want to acknowledge members of the **COS2MOS (Computer Science for Complex System modelling)** research group that have been involved on referenced works: **Unai Arronategui**, **José Manuel Colom**, **Victor Medel**, **Rafael Tolosana**, **Omer F. Rana** (University of Cardiff) and **Congduc Pham** (Université de Pau et des Pays de l'Adour).

References

1. Laney, D.: 3D data management: Controlling data volume, velocity, and variety. Technical report, META Group (February 2001)
2. Kim, H., Khamra, Y.E., Jha, S., Parashar, M.: An autonomic approach to integrated HPC grid and cloud usage. In: Fifth International Conference on e-Science, E-SCIENCE 2009, Oxford, UK, IEEE Computer Society (December 2009) 366–373
3. Fehling, C., Leymann, F., Retter, R., Schupeck, W., Arbitter, P.: Cloud Computing Patterns - Fundamentals to Design, Build, and Manage Cloud Applications. Springer (2014)
4. Yeo, S., Lee, H.H.: Using mathematical modeling in provisioning a heterogeneous cloud computing environment. Computer **44**(8) (2011) 55–62
5. O'Loughlin, J., Gillam, L.: Performance evaluation for cost-efficient public infrastructure cloud use. In: Economics of Grids, Clouds, Systems, and Services - 11th International Conference, GECON'14 , Cardiff, UK, September 16-18, 2014. Volume 8914 of LNCS. (2014) 133–145
6. Bañares, J.Á., Tolosana-Calasanz, R., Tricas, F., Arronategui, U., Celaya, J., Colom, J.M.: Construction of data streams applications from functional, non-functional and resource requirements for electric vehicle aggregators. The COSMOS vision. In: International Workshop on Petri Nets and Software Engineering, PNSE 2014. Number 1160 in CEUR Workshop Proceedings, Aachen, CEUR-WS.org (2014) 333–334 http://ceur-ws.org/Vol-1160/.
7. Tolosana-Calasanz, R., Bañares, J.Á., Colom, J.M.: Towards Petri net-based economical analysis for streaming applications executed over cloud infrastructures. In: Economics of Grids, Clouds, Systems, and Services - 11th International Conference, GECON'14, Cardiff, UK, September 16-18, 2014. Volume 8914 of LNCS. (2014) 189–205
8. Tolosana-Calasanz, R., Bañares, J.Á., Rana, O., Pham, C., Xydas, E., Marmaras, C., Papadopoulos, P., Cipcigan, L.: Enforcing quality of service on opennebula-based shared clouds. In: CCGRID'14 Workshop on Data-intensive Process Management in Large-Scale Sensor Systems, DPMSS'14, Chicago, IL, USA, May 26-29, 2014, IEEE (2014) 651–659
9. Merino, A., Tolosana-Calasanz, R., Bañares, J.Á., Colom, J.M.: A specification language for performance and economical analysis of short term data intensive energy management services. In: Economics of Grids, Clouds, Systems, and Services - 12th International Conference, GECON'15, Cluj-Napoca, Romania, September 15-17, 2015. Volume 9512 of LNCS. (2015) 1–17

10. Etzion, O., Niblett, P.: Event Processing in Action. 1st edn. Manning Publications Co., Greenwich, CT, USA (2010)
11. Pautasso, C., Alonso, G.: Parallel computing patterns for Grid workflows. In: Proceedings of the HPDC 2006 Workshop on Workflows in Support of Large-Scale Science, WORKS 2006, June 19-23, Paris, France. (2006) 1–10
12. Yu, L., Moretti, C., Thrasher, A., Emrich, S.J., Judd, K., Thain, D.: Harnessing parallelism in multicore clusters with the All-Pairs, Wavefront, and Makeflow abstractions. Cluster Computing **13**(3) (2010) 243–256
13. Simmhan, Y., Kumbhare, A.G.: Floe: A continuous dataflow framework for dynamic cloud applications. CoRR **abs/1406.5977** (2014)
14. Lohrmann, B., Janacik, P., Kao, O.: Elastic stream processing with latency guarantees. In: Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on. (June 2015) 399–410
15. Seceleanu, C.C., Crnkovic, I.: Component models for reasoning. IEEE Computer **46**(11) (2013) 40–47
16. Tolosana-Calasanz, R., Bañares, J.Á., Colom, J.M.: On autonomic platform-as-a-service: Characterisation and conceptual model. In Jezic, G., Howlett, R.J., Jain, L.C., eds.: Agent and Multi-Agent Systems: Technologies and Applications. Volume 38 of Smart Innovation, Systems and Technologies. Springer International Publishing (2015) 217–226
17. Murata, T.: Petri nets: Properties, analysis and applications. In: Proceedings of IEEE. Volume 77. (April 1989) 541–580
18. Campos, J., Chiola, G., Colom, J.M., Silva, M.: Properties and performance bounds for timed marked graphs. Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on **39**(5) (1992) 386–401
19. Harrison, P.G., Patel, N.M.: Performance Modelling of Communication Networks and Computer Architectures. Addison-Wesley Longman Publishing Co., Inc. (1992)
20. Khazaei, H., Misic, J., Misic, V.: Performance analysis of cloud computing centers using m/g/m/m+r queuing systems. IEEE Transactions on Parallel and Distributed Systems **23**(5) (2012) 936–943
21. Schad, J., Dittrich, J., Quiané-Ruiz, J.A.: Runtime measurements in the cloud: Observing, analyzing, and reducing variance. Proc. VLDB Endow. **3**(1-2) (September 2010) 460–471
22. Weingärtner, R., Bräscher, G.B., Westphall, C.B.: Cloud resource management: A survey on forecasting and profiling models. J. Network and Computer Applications **47** (2015) 99–106
23. Medel, V., Rana, O., Bañares, J.a., Arronategui, U.: Modelling performance and resource management in kubernetes. In: Proceedings of the 9th International Conference on Utility and Cloud Computing. UCC '16, New York, NY, USA, ACM (2016) 257–262
24. Medel, V., Rana, O., Bañares, J.Á., Arronategui, U.: Adaptive application scheduling under interference in kubernetes. In: Proceedings of the 9th International Conference on Utility and Cloud Computing, ACM (2016) 426–427
25. Tolosana-Calasanz, R., BañAres, J.Á., Pham, C., Rana, O.F.: Enforcing qos in scientific workflow systems enacted over cloud infrastructures. Journal of Computer and System Sciences **78**(5) (2012) 1300–1315
26. Tolosana-Calasanz, R., Bañares, J.A., Rana, O.F.: Autonomic streaming pipeline for scientific workflows. Concurrency and Computation: Practice and Experience **23**(16) (2011) 1868–1892

27. Tolosana-Calasanz, R., Bañares, J.Á., Pham, C., Rana, O.F.: Resource management for bursty streams on multi-tenancy cloud environments. Future Generation Computer Systems **55** (2016) 444–459
28. Tolosana-Calasanz, R., Rana, O.F., Bañares, J.A.: Automating performance analysis from Taverna workflows. In: Component-Based Software Engineering: 11th International Symposium, CBSE 2008, Karlsruhe, Germany, October 14-17, 2008. Proceedings. (2008) 1–15
29. Tolosana-Calasanz, R., Bañares, J.A., Rana, O.F., Álvarez, P., Ezpeleta, J., Hoheisel, A.: Adaptive exception handling for scientific workflows. Concurr. Comput. : Pract. Exper. **22**(5) (April 2010) 617–642
30. Yi, S., Li, C., Li, Q.: A survey of fog computing: concepts, applications and issues. In: Proceedings of the 2015 Workshop on Mobile Big Data, ACM (2015) 37–42
31. Tolosana-Calasanz, R., Bañares, J.Á., Pham, C., Rana, O.F.: Resource management for bursty streams on multi-tenancy cloud environments. Future Generation Computer Systems **55** (2016) 444 – 459

Verification of Reconfigurable Petri Nets

Julia Padberg

HAW Hamburg, Department of Informatics,
http://users.informatik.haw-hamburg.de/~padberg/

Abstract We introduce a family of modeling techniques consisting of Petri nets together with a set of rules. For reconfigurable Petri nets, e.g. in [3] not only the follower marking can be computed but also the structure can be changed by rule application to obtain a new net. Motivation is the observation that in increasingly many application areas the underlying system has to be dynamic in a structural sense. Complex coordination and structural adaptation at run-time (e.g. mobile ad-hoc networks, dynamic hardware reconfiguration, communication spaces, ubiquitous computing) are main features that need to be modelled adequately. The distinction between the net behaviour and the dynamic change of its net structure is the characteristic feature that makes reconfigurable Petri nets so suitable for modeling systems with dynamic structures.

For rule-based modification of Petri nets we use the framework of net transformations that is inspired by graph transformation systems [2]. The basic idea behind net transformation is the stepwise modification of Petri nets by given rules. The rules present a rewriting of nets where the left-hand side is replaced by the right-hand side. The abstract semantics we introduce in [4] is a graph with nodes that consist of isomorphism classes of the net structure and an isomorphism class of the current marking. Arcs between these nodes represent computation steps being either a transition firing or a direct transformation. Based on this semantics we can define properties and model-check these properties.

Model checking is a widely used technique to prove properties such as liveness, deadlock or safety for a given model. Here we present model checking of reconfigurable Petri nets [7,6]. The main task is to flatten the two levels of dynamic behavior that reconfigurable nets provide, the firing of transitions on the one hand and the transformation of the nets on the other hand. We show how to translate a reconfigurable net into Maude modules [1]. Maude's LTL model-checker is then used to verify properties of these modules. The correctness of this conversion is proven as the corresponding labelled transitions systems are bisimular.

In an ongoing example reconfigurable Petri nets are used to model and to verify partial dynamic reconfiguration of field programmable gate arrays using the tool `ReconNet` ([5] or see https://reconnetblog.wordpress.com/).

Keywords: Petri Nets, Verification, Reconfigurable Petri Nets

References

1. Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Jose F. Quesada. Maude: specification and programming in rewriting logic. *Theor. Comput. Sci.*, 285(2):187–243, 2002.
2. H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in TCS. Springer, 2006.
3. Hartmut Ehrig and Julia Padberg. Graph grammars and Petri net transformations. In *Lectures on Concurrency and Petri Nets*, pages 496–536, 2004.
4. J. Padberg. Abstract interleaving semantics for reconfigurable Petri nets. *ECE-ASST*, 51, 2012.
5. Julia Padberg, Marvin Ede, Gerhard Oelker, and Kathrin Hoffmann. Reconnet: A tool for modeling and simulating with reconfigurable place/transition nets. *ECE-ASST*, 54, 2012.
6. Julia Padberg and Alexander Schulz. Model checking reconfigurable Petri nets with Maude. In Rachid Echahed and Mark Minas, editors, *Graph Transformation, 9th Int. Conf. on*, volume 9761 of *Lecture Notes in Computer Science*, pages 54–70. Springer, 2016.
7. Alexander Schulz. Model checking of reconfigurable Petri nets. Master's thesis, University of Applied Sciences Hamburg, 2015.

Part II

Long Presentations

Decision Diagrams for Petri Nets: which Variable Ordering?

Elvio Gilberto Amparore[1], Susanna Donatelli[1], Marco Beccuti[1], Giulio Garbi[1], Andrew Miner[2]

[1] Università di Torino, Dipartimento di Informatica, Italy
{amparore,susi,beccuti}@di.unito.it
[2] Iowa State University, Iowa, USA
asminer@iastate.edu

Abstract. The efficacy of decision diagram techniques for state space generation is known to be heavily dependent on the variable order. Ordering can be done a-priori (static) or during the state space generation (dynamic). We focus our attention on static ordering techniques. Many static decision diagram variable ordering techniques exist, but it is hard to choose which method to use, since only fragmented information about their performance is available. In the work reported in this paper we used the models of the Model Checking Contest 2016 edition to provide an extensive comparison of 14 different algorithms, in order to better understand their efficacy. Comparison is based on the size of the decision diagram of the generated state space. The state space is generated using the Saturation method provided by the Meddly library.

Keywords: decision diagrams, static variable ordering, heuristic optimization, saturation.

1 Introduction

Binary Decision diagram (BDD) [10] is a well-known data structure that was extensively used in industrial hardware verification thanks to its ability of encoding complex boolean functions on very large domains. In the context of discrete event dynamic systems in general, and of Petri nets in particular, BDDs and its extensions (e.g. Multi-way Decision Diagram -MDD) were proposed to efficiently generate and store the state space of complex systems. Indeed, symbolic state space generation techniques exploit Decision Diagrams (DDs) because they allow to encode and manipulate entire sets of states at once, instead of storing and exploring each state explicitly.

The size of DD representation is known to be strongly dependent on variable orders: a good ordering can significantly change the memory consumption and the execution time needed to generate and encode the state space of a system. Unfortunately finding the optimal variable ordering is known to be NP-complete [9]. Therefore, efficient DD generation is usually reliant on various heuristics for the selection of (sub)optimal orderings. In this paper we will only

consider *static* variable ordering, i.e. once the variable ordering l is selected, the MDD construction starts without the possibility of changing l. In the literature several papers were published to study the topic of variable ordering. An overview of these works can be found in [25], and in the recent work in [19]. In particular the latter work considers a new set of variable ordering algorithms, based on Bandwidth-reduction methods [27], and observes that they can be successfully applied for variable ordering. We also consider the work published in [18] (based on the ideas in [28]), which are state-of-the-art variable ordering methods specialized for Petri nets.

The motivation of this work was to understand how these different algorithms for variable orderings behave. Also, we wanted to investigate whether the availability of structural information on the Petri net model could make a difference. As far as we know there is no extensive comparison of these specific methods.

In particular we have addressed the following research objectives:

1. Build an environment (a *benchmark*) in which different algorithms can be checked on a vast number of models.
2. Investigate whether structural information like P-semiflows can be exploited to define better algorithms for variable orderings.
3. What are the right metrics to compare variable ordering algorithms in the most fair manner.

To achieve these objectives we have built a benchmark in which 14 different algorithms for variable orderings have been implemented and compared on state space generation of the Petri nets taken from the models of the Model Checking context (both colored and uncolored), 2016 edition [16]. The implementation is part of RGMEDD [6], the model-checker of GreatSPN [5], and uses MDD saturation [12]. The ordering algorithms are either taken from the literature (both in their basic form and with a few new variations) or they were already included in GreatSPN.

Figure 1, left, depicts the benchmark workflow. Given a net system $\mathcal{S} = (\mathcal{N}, m_0)$ all ordering algorithms in $\mathcal{A}$ are run (box 1), then the reachability set RS_l of the system is computed *for each* ordering $l \in \mathcal{L}$ (box 2) and algorithms are ranked according to some MDD metrics $\mathrm{MM}(RS_l)$, (box 3). The best algorithm a^* is then the best algorithm for solving the PN system $\mathcal{S} = (\mathcal{N}, m_0)$ (box 4) and its state space RS_l could be the one used to check properties.

This workflow allows to: 1) provide indications on the best performing algorithm for a given model and 2) compare the algorithms in $\mathcal{A}$ on a large set of models to possibly identify the algorithm with the best average performances. The problem of defining a ranking among algorithms (or of identifying the "best" algorithm) is non-trivial and will be explored in Section 3.

Figure 1, right, shows a high level view of the approach used to compare variable ordering algorithms in the benchmark. Columns represents algorithms, and rows represents *model instances*, that is to say a Petri net model with an associated initial marking. A square in position (j, k) represents the state space generation for the jth model instance done by GreatSPN using the variable

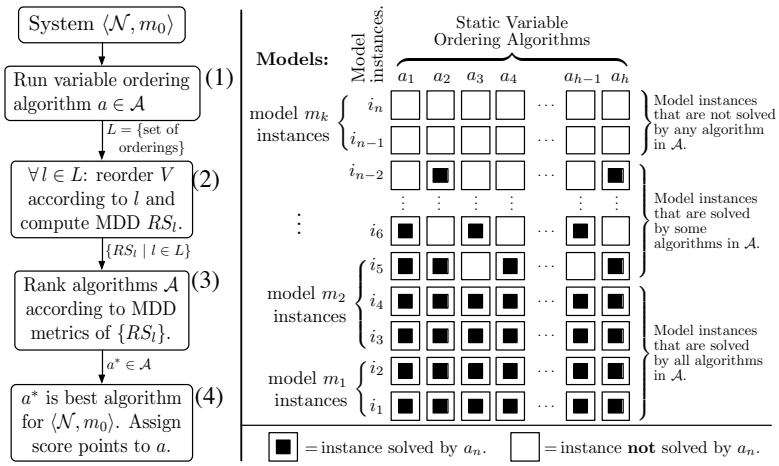


Fig. 1. Workflow for analysis and testing of static variable ordering algorithms.

ordering computed by algorithm a_k. A black square indicates that the state space was generated within the given time and memory limits.

In the analysis of the results from the benchmark we shall distinguish among model instances for which no variable ordering was good enough to allow GreatSPN to generate the state space (only white squares on the model instance row, as for the first two rows in the figure), model instances for which at least one variable ordering was good enough to generate the state space (at least one black square in the row), and model instances in which GreatSPN generates the state space with all variable orderings (all black squares in the row), that we shall consider "easy" instances.

In the analysis it is also important to distinguish whether we evaluate ordering algorithms w.r.t. all possible instances or on a representative set of them. Figure 1, right, highlights that instances are not independent, since they are often generated from the same "model" that is to say the same Petri net $\mathcal{N}$ by varying the initial marking m_0 or some other parameter (like the cardinality of the color classes). As we shall see in the experimental part, collecting measures over all instances, in which all instances have the same weight, may lead to a distortion of the observed behaviour, since the number of instances per model can differ significantly. A measure "per model" is therefore also considered.

This work could not have been possible without the models made available by the Model Checking Contest, the functions of the Meddly MDD library and the GreatSPN framework. We shall now review them in the following.

Model Checking Contest. The Model Checking Contest[16] is a yearly scientific event whose aim is to provide a comparison among the different available verification tools. The 2016 edition employed a set of 665 PNML instances generated from 65 (un)colored models, provided by the scientific community. The participating tools are compared on several examination goals, i.e. state space,

reachability, LTL and CTL formulas. The MCC team has designed a score system to evaluate tools that we shall employ as one of the considered benchmark metrics for evaluating the algorithms, as evaluating the orderings can be reduced to evaluating the same tool, GreatSPN, in as many variations as the number of ordering algorithms considered.

Meddly library. Meddly (Multi-terminal and Edge-valued Decision Diagram LibrarY) [8] is an open-source library implementation of Binary Decision Diagrams (BDDs) and several variants, including Multi-way Decision Diagrams (MDDs, implemented "natively") and Matrix Diagrams (MxDs, implemented as MDDs with an identity reduction rule). Users can construct one or more forests (collections of nodes) over the same or different domains (collections of variables). Several "apply" operations are implemented, including customized and efficient relational product operations and saturation [12] for generating the set of states (as an MDD) reachable from an initial set of states according to a transition relation (as an MxD). Saturation may be invoked either with an already known ("pre-generated") transition relation, or with a transition relation that is built "on the fly" during saturation [13], although this is currently a prototype implementation. The transition relation may be specified as a single monolithic relation that is then automatically split [23], or as a relation partitioned by levels or by events [14], which is usually preferred since the relation for a single Petri net transition tends to be small and easy to construct.

GreatSPN framework. GreatSPN is a well-known collection of tools for the design and analysis of Petri net models [5, 6]. The tools are aimed at the qualitative and quantitative analysis of Generalized Stochastic Petri Net (GSPN) [1] and Stochastic Symmetrical Net (SSN) through computation of structural properties, state space generation and analysis, analytical computation of performance indices, fluid approximation and diffusion approximation, symbolic CTL model checking, all available through a common graphical user interface [4]. The state space generation [7] of GreatSPN is based on Meddly. In this paper we use the collection of variable ordering heuristics implemented in GreatSPN. This collection has been enlarged to include all the variable ordering algorithms described in Section 2.

The paper is organized as follows: Section 2 reviews the considered algorithms; Section 3 describes the benchmark (models, scores and results); Section 4 considers a parameter estimation problem for optimizing one of the best methods found (Sloan) and Section 5 concludes the paper outlining possible future directions of work.

2 The set $\mathcal{A}$ of variable ordering algorithms

In this section we briefly review the algorithms considered by the benchmark. Although our target model is that of Petri nets, we describe the algorithms in a more general form (as some of them were not defined specifically for Petri nets). We therefore consider the following high level description of the model.

Let V be the set of *variables*, that translates directly to MDD levels. Let E be the set of events in the model. Events are connected to *input* and *output* variables. Let $V^{\text{in}}(e)$ and $V^{\text{out}}(e)$ be the sets of input and output variables of event e, respectively. Let $E^{\text{in}}(v)$ and $E^{\text{out}}(v)$ be the set of events to which variable v participates as an input or as an output variable, respectively. For some models, structural information is known in the form of disjoint variables partitions named *structural units*. Let Π be the set of structural units. With $\Pi(v)$ we denote the unit of variable v. Let $V(\pi)$ be the set of vertices in unit $\pi \in \Pi$. In this paper we consider three different types of structural unit sets. Let Π_{PSF} be the set of units corresponding to the P-semiflows of the net, obtained ignoring the place multiplicity. On some models, structural units are known because they are obtained from the composition of smaller parts. We mainly refer to [17] for the concept of Nested Units (NU). Let Π_{NU} be this set of structural units, which is defined only for a subset of models. Finally, structural units can be derived using a clustering algorithm. Let Π_{Cl} be such set of units. We will discuss clustering in section 2.5.

Following this criteria, we subdivide the set of algorithms $\mathcal{A}$ into: The set $\mathcal{A}_{\text{Gen}}$, that do not use any structural information; The set $\mathcal{A}_{\text{PSF}}$ that requires Π_{PSF}; The set $\mathcal{A}_{\text{NU}}$ that require Π_{NU}. Since clustering can be computed on almost any model, we consider method that use Π_{Cl} as part of $\mathcal{A}_{\text{Gen}}$.

In our context, the set of MDD variables V corresponds to the place set of the Petri net, and the set of events is the transition set of the Petri net. Let $l : V \to \mathbb{N}$ be a variable order, i.e. an assignment of consecutive integer values to the variables V.

2.1 Force-based orderings

The FORCE heuristic, introduced in [3], is a n-dimensional graph layering technique based on the idea that variables form an hyper-graph, such that variables connected by the same event are subject to an "attractive" force, while variables not directly connected by an event are subject to a "repulsive" force. It is a simple method that can be summarized as follows:

Algorithm 1 Pseudocode of the FORCE heuristic.

Function `FORCE`:
  Shuffle the variables randomly.
  repeat K times:
    for each event $e \in E$:
      compute *center of gravity* $cog_e = \frac{1}{|e|} \sum_{v \in e} l(v)$
    for each variable $v \in V$:
      compute hyper-position $p(v) = \frac{1}{E(v)} \sum_{e \in E(v)} cog_e$
    Sort vertices according the their $p(v)$ value.
    Weight obtained variable order using metric function $f(l)$.
  return the variable order that had the best metric among
      the K generated permutations.

Algorithm 1 gives the general skeleton of the FORCE algorithm. One of the most interesting part of this method is that it can be seen as a *factory* of variable orders, that generates K different orders. The algorithm starts by shuffling the variables set, then it iterates K times trying to achieve a convergence. The version of FORCE that we tested uses $K = 200$. In our experience, FORCE does not converge stably to a single permutation on most models, and it generates a new permutation at every iteration.

After having produced a candidate variable order l, a metric function $f(l) \to \mathbb{R}$ is used to estimate the *quality* of that order. The chosen order is then the one that maximises (or minimises) the target metric function. In our benchmark we tested the following metric functions:

- *Point-Transition Spans*: the PTS score is given by the sum of all distances between the center of gravities cog_e and their connected variables. The formula is the following: $\mathrm{PTS}(l) = \frac{1}{|E|\cdot|V|} \sum_{e\in E} \left(\sum_{v\in V(e)} \left| p(v) - cog_e \right| \right)$.
- *Normalized Event Span*: the NES score [26] is based on the idea of measuring the sum of the *event spans* of each event e. The event span $span_l(e)$ of event e for the variable order l is defined as the largest distance in l between every pair of variables $v, v' \in V(e)$. The metric is then defined as a normalized sum of these spans: $\mathrm{NES}(l) = \frac{1}{|E|} \sum_{e\in E} \frac{span(e)+1}{|V|}$.
- *Weighted Event Span*: WES [26] is a modified version of NES where events closer to the top of the variable order weigh more. This modification is based on the assumption that the Saturation algorithm performs better when most events are close to the bottom of the order. The formula of WES^i is: $\mathrm{WES}^i(l) = \frac{1}{|E|} \sum_{e\in E} \frac{span(e)+1}{|V|} \cdot \left(\frac{2\cdot top(e)}{|V|} \right)^i$, with $i = 1$.

In addition to the evaluation metrics, structural information of the model can be used to establish additional *center of gravity*. We tested three variations of the FORCE method, which are:

- `FORCE-*`: Events are used as centers of gravity, as described in Algorithm 1, where `*` is one among `PTS`, `NES` or `WES`.
- `FORCE-P`: P-semiflows are used as center of gravities instead of the events. The method is only tested for those models that have P-semiflows. It uses the WES metric to select between the K generated orderings.
- `FORCE-NU`: Structural units are used as center of gravities, along with the events. This intuitively tries to keep together those variables that belong to the same structural unit. Again, WES is used for the metric function.

The set $\mathcal{A}$ of algorithms considered in the benchmark includes: `FORCE-PTS`, `FORCE-NES` and `FORCE-WES` in $\mathcal{A}_{\mathrm{Gen}}$; the method `FORCE-P` in $\mathcal{A}_{\mathrm{PSF}}$; and the method `FORCE-NU` in $\mathcal{A}_{\mathrm{NU}}$, for a total of 5 variations of this method.

2.2 Bandwidth-reduction methods

Bandwidth-reduction(BR) methods are a class of algorithms that try to permute a sparse matrix into a band matrix, i.e. where most of the non-zero entries are

confined in a (hopefully) small band near the diagonal. It is known [11] that reducing the event span in the variable order generally improves the compactness of the generated MDD. Therefore, event span reduction can be seen as an equivalent of a bandwidth reduction on a sparse matrix. A first connection between bandwidth-reduction methods and variable ordering has been tested in [19] and in [22] on the model bipartite graph. In these works the considered BR methods where: The Reverse Cuthill-Mckee [15]; The King algorithm [20]; and The Sloan algorithm [27]. The choice was motivated by their ready availability in the Boost-C++ and in the ViennaCL library. In particular, Sloan, which is the state-of-the-art method for bandwidth reduction, showed promising performances as a variable ordering method. In addition, Sloan almost always outperforms [19] the other two BR methods, but for completeness of our benchmark we have decided to replicate the results. We concentrate our review on the Sloan method only, due to its effectiveness and its parametric nature.

The goal of the Sloan algorithm is to condensate the entries of a symmetric square matrix $\mathbf{A}$ around its diagonal, thus reducing the matrix *bandwidth* and *profile* [27]. It works on symmetric matrices only, hence it is necessary to impose some form of translation of the model graph into a form that is accepted by the algorithm. The work in [22] adopts the symmetrization of the dependency graph of the model, i.e. the input matrix $\mathbf{A}$ for the Sloan algorithm will have $(|V|+|E|)\times(|V|+|E|)$ entries. We follow instead a different approach. The size of $\mathbf{A}$ is U, with $|V| \leq U \leq |V|+|E|$. Every event e generates entries in $\mathbf{A}$: when $|V^{\text{in}}(e)|\times|V^{\text{out}}(e)| < T$, with T a threshold value, all the cross-product of $V^{\text{in}}(e)$ vertices with the $V^{\text{out}}(e)$ vertices are nonzero entries in $\mathbf{A}$. If instead $|V^{\text{in}}(e)|\times|V^{\text{out}}(e)| \geq T$, a pseudo-vertex v_e is added, and all $V^{\text{in}}(e)\times\{v_e\}$ and $\{v_e\}\times V^{\text{out}}(e)$ entries in $\mathbf{A}$ are set to be nonzero. Usually U will be equal to V, or just slightly larger. This choice better represents the variable–variable interaction matrix, while avoiding degenerate cases where highly connected event could generate a dense matrix $\mathbf{A}$. In our implementation, T is set to 100. The matrix is finally made symmetric using: $\mathbf{A}' = \mathbf{A} + \mathbf{A}^T$. As we shall see in section 3, the computational cost of Sloan remains almost always bounded.

Algorithm 2 Pseudocode of the Sloan algorithm.

Function `Sloan`:
- Select a vertex u of the graph.
- Select v as the most-distant vertex to u with a graph visit.
- Establish a gradient from 0 in v to d in u using a depth-first visit.
- Initialize visit frontier $Q = \{v\}$
- **repeat** until Q is empty:
  - Remove from the frontier Q the vertex v' that minimizes $P(v')$.
  - Add v' to the variable ordering l.
  - Add the unexplored adjacent vertices of v' to Q.

A second relevant characteristic of Sloan is its *parametric priority function* $P(v')$, which guides variable selection in the greedy strategy. A very compact pseudocode of Sloan is given in Algorithm 2. A more detailed one can be found in [21]. The method follows two phases. In the first phase it searches a pseudo-diameter of the $\mathbf{A}$ matrix graph, i.e. two vertices v, u that have an (almost) maximal distance. Usually, an heuristic approach based on the construction of the *root level structure* of the graph is employed. The method then performs a visit, starting from v, exploring in sequence all vertices in the visit frontier Q that maximize the priority function:

$$P(v') = -W_1 \cdot incr(v') + W_2 \cdot dist(v, v')$$

where $incr(v')$ is the number of unexplored vertices adjacent to v', $dist(v, v')$ is the distance between the initial vertex v and v', and W_1 and W_2 are two integer weights. The weights control how Sloan prioritises the visit of the local cluster (W_1) and how much the selection should follow the gradient (W_2). Since the two weights control a linear combination of factors, in our analysis we shall consider only the ratio $\frac{W_1}{W_2}$. In Section 4 we will concentrate on the best selection of this ratio. We use the name `SLO`, `CM` and `KING` to refer to the Sloan, Cuthill-Mckee and King algorithms in the $\mathcal{A}_{\text{Gen}}$ set. GreatSPN uses the Boost-C++ implementations of these methods.

2.3 P-semiflows chaining algorithm

In this subsection we propose a new heuristic algorithm exploiting the Π_{PSF} set of structural units obtained by the P-semiflows computation. A P-semiflow is a positive, integer, left annuler of the incidence matrix of a Petri net, and it is known that, in any reachable marking, the sum of tokens in the net places, weighted by the P-semi-flow coefficients, is constant and equal to the weighted sum of the initial marking (P-invariant). Its main idea is to maintain the places shared between two Π_{PSF} units (i.e. P-semiflows) as close as possible in the final MDD variable ordering, since their markings cannot vary arbitrarily. The pseudo-code is reported in Algorithm 3. The algorithm takes as input the Π_{PSF} set and returns as output a variable ordering (stored in the ordered l). Initially, the π_i unit sharing the highest number of places with another unit is removed by Π_{PSF} and saved in π_{curr}. All its places are added to l.

Then the main loop runs until Π_{PSF} becomes empty. The loop comprises the following operations. The π_j unit sharing the highest number of places with π_{curr} is selected. All the places of π_j in l, which are not currently C (i.e. the list of currently discovered common places) are removed. The common places between π_i and π_j not present in C are appended to l. Then the places present only in π_j are added to l. After these steps, C is updated with the common places in π_i and π_j, and π_j is removed by Π_{PSF}. Finally π_{curr} becomes π_j, completing the iteration. This algorithm is named `P` and belongs to the $\mathcal{A}_{\text{PSF}}$ set.

Algorithm 3 Pseudocode of the P-semiflows chaining algorithm.

Function `P-semiflows(`Π_{PSF}`)`:
  $l = \varnothing$ is the ordered list of places.
  $C = \varnothing$ is the set of current discovered common places.
  Select a unit $\pi_i \in \Pi_{\text{PSF}}$ s.t. $\max_{\{i,j\} \in |\Pi_{\text{PSF}}|} \pi_i \cap \pi_j$ with $i \neq j$
  $\Pi_{\text{PSF}} = \Pi_{\text{PSF}} \setminus \{\pi_i\}$
  $\pi_{curr} = \pi_i$
  Append $V(\pi_{curr})$ to l
  repeat until Π_{PSF} is empty:
    Select a unit $\pi_j \in \Pi_{\text{PSF}}$ s.t. $\max_{j \in |\Pi_{\text{PSF}}|} \pi_{curr} \cap \pi_j$
    Remove $(l \cap V(\pi_j)) \setminus C$ to l
    Append $V(\pi_{curr} \cap \pi_j) \setminus C$ to l
    Append $V(\pi_j) \setminus (C \cap V(\pi_{curr}))$ to l
    Add $V(\pi_{curr} \cap \pi_j)$ to C
    $\pi_{curr} = \pi_j$
    $\Pi_{\text{PSF}} = \Pi_{\text{PSF}} \setminus \{\pi_j\}$
  return l

2.4 The Noack and the Tovchigrechko greedy heuristics algorithms

The Noack [24] and the Tovchigrechko [28] methods are greedy heuristics that build up the variable order sequence by picking, at every iteration, the variable that minimizes an objective function. A detailed description can be found in [18]. A pseudo-code is given in Algorithm 4.

Algorithm 4 Pseudocode of the Noack/Tovchigrechko heuristics.

Function `NOACK-TOV`:
  $S = \varnothing$ is the set of already selected places.
  for i from 1 to $|V|$:
    compute weight $W(v) = f(v, S)$ for each $v \notin S$.
    find v that maximizes $W(v)$.
    $l(i) = v$.
    $S \leftarrow S \cup \{v\}$.
  return the variable order l.

The main difference between the Noack and the Tovchigrechko methods is the weight function $f(v, S)$, defined as:

$$f_{\text{Noack}}(v, S) = \sum_{\substack{e \in E^{\text{out}}(v) \\ k_1(e) \wedge k_2(e)}} \big(g_1(e) + z_1(e)\big) + \sum_{\substack{e \in E^{\text{in}}(v) \\ k_1(e) \wedge k_2(e)}} \big(g_2(e) + c_2(e)\big)$$

$$f_{\text{Tov}}(v, S) = \sum_{\substack{e \in E^{\text{out}}(v) \\ k_1(e)}} g_1(e) + \sum_{\substack{e \in E^{\text{out}}(v) \\ k_2(e)}} c_1(e) + \sum_{\substack{e \in E^{\text{in}}(v) \\ k_1(e)}} g_2(e) + \sum_{\substack{e \in E^{\text{in}}(v) \\ k_2(e)}} c_2(e)$$

where the sub-terms are defined as:

$$g_1(e) = \frac{\max\left(0.1,\ |S \cap V^{\text{in}}(e)|\right)}{|V^{\text{in}}(e)|}, \qquad g_2(e) = \frac{1+|S \cap V^{\text{in}}(e)|}{|V^{\text{in}}(e)|}$$
$$c_1(e) = \frac{\max\left(0.1,\ 2\cdot|S \cap V^{\text{out}}(e)|\right)}{|V^{\text{out}}(e)|}, \qquad c_2(e) = \frac{\max\left(0.2,\ 2\cdot|S \cap V^{\text{out}}(e)|\right)}{|V^{\text{out}}(e)|}$$
$$z_1(e) = \frac{2\cdot|S \cap V^{\text{out}}(e)|}{|V^{\text{out}}(e)|}, \quad k_1(e) = |V^{\text{in}}(e)| > 0, \quad k_2(e) = |V^{\text{out}}(e)| > 0$$

Few technical information is known about the criteria that were followed for the definition of the f_{Noack} and f_{Tov} functions. An important characteristic is that both functions have different criteria for input and output event conditions, i.e. they do not work on the symmetrized event relation, like the Sloan method.

The set $\mathcal{A}$ of algorithms considered in the benchmark includes four variations of these two algorithms: the variants `NOACK` and `TOV` are the standard implementations, as described in [18]. In addition, we tested whether these methods could benefit from an additional weight function that favours structural units. The two methods `NOACK-NU` and `TOV-NU` employ this technique. In this case, function $f(v, S)$ is modified to add a weight of $0.75 \cdot |S \cap V(\Pi(v))|$ to variable v. In our experiments we tried various values for the weight, but we generally observed that these modified methods do not benefit from this additional information.

2.5 Markov Clustering heuristic

The heuristic `MCL` is based on the idea of exploring the effectiveness of clustering algorithms to improve variable order technique. The hypothesis is that in some models, it could be beneficial to first group places that belong to connected clusters. For our tests we selected the Markov Cluster algorithm [29]. The method works as a modified version of Sloan, were clusters are first ordered according to their average gradient, and then places belonging to the same clusters will appear consecutively on the variable ordering, following the cluster orders. This method is named `MCL` and belongs to the $\mathcal{A}_{\text{Gen}}$ set.

3 The Benchmark

The considered model instances are that of the Model Checking Contest, 2016 edition [16], which consists of 665 PNML models. We discarded 257 instances that our tool was not capable to solve in the imposed time and memory limits, because either the net was too big or the RS MDD was too large under any considered ordering. Thus, we considered for the benchmark the set $\mathcal{I}$ made of 386 instances, belonging to a set $\mathcal{M}$ of 62 models. These 386 instances run for the 14 tested algorithms for 75 minutes, with 16GB of memory and a decision diagram cache of 2^{26} entries. In the 386 instances of $\mathcal{I}$ two sub-groups are identified: The set $\mathcal{I}_{\text{PSF}} \subset \mathcal{I}$ of instances for which P-semiflows are available, with 328 instances generated from a subset $\mathcal{M}_{\text{PSF}}$ of 55 models; The set $\mathcal{I}_{\text{NU}} \subset \mathcal{I}$ of instances for which nested units are available, with 65 instances generated from a subset $\mathcal{M}_{\text{NU}}$ of 15 models.

The overall tests were performed on OCCAM [2] (Open Computing Cluster for Advanced data Manipulation), a multi-purpose flexible HPC cluster designed and maintained by a collaboration between the University of Torino and the Torino branch of the National Institute for Nuclear Physics. OCCAM counts slightly more than 1100 cpu cores including three different types of computing nodes: standard Xeon E5 dual-socket nodes, large Xeon E5 quad-sockets nodes with 768 GB RAM, and multi-GPU NVIDIA nodes. Our experiments took 221 hours of computation using 3 standard Xeon E5 dual-socket nodes.

Scores. Typically, the most important parameter that measures the performance of variable ordering is the MDD peak size, measured in nodes. The peak size represents the maximum MDD size reached during the computation, and it therefore represents the memory requirement. It is also directly related to the time cost of building the MDD. For these reasons we preferred to use the peak size alone rather than a weighted measure of time, memory and peak size, that would make the result interpretation more complex. The peak size is, however, a quantity that is strictly related to the model instance. Different instances will have unrelated peak sizes, often with different magnitudes. To treat instances in a balanced way, some form of normalized score is needed. We consider three different score functions: for all of them the score of an algorithm is first normalized against the other algorithms on the same instance, which gives a score per instance, and then summed over all instances. Let i be an instance, solved by algorithms $\mathcal{A} = \{a_1, \ldots, a_m\}$ with peak nodes $P_i = \{p_{a_1}(i), \ldots, p_{a_m}(i)\}$. The scores of an *algorithm a for an instance i* are:

- The *Mean Standard Score of instance i* is defined as: $\mathrm{MSS}_a(i) = \frac{p_a(i) - \mu_{\mathcal{A}}(i)}{\sigma_{\mathcal{A}}(i)}$, where $\mu_{\mathcal{A}}(i)$ and $\sigma_{\mathcal{A}}(i)$ are the mean and standard deviations for instance summed over the all algorithms that solve instance i.
- The *Normalized Score for instance i* is defined as: $\mathrm{NS}_a(i) = 1 - \frac{\min\{p \in P_i\}}{p_a(i)}$, which just rescales the peak nodes taking the minimum as the scaling factor.
- The *Model Checking Contest score*[1] *for instance i* defined as: $\mathrm{MCC}_a(i) = 48$ if a terminates on i in the given time bound, plus 24 if $p_a(i) = \min\{p \in P_i\}$.

The final scores used for ranking algorithms over a set $\mathcal{I}' \subseteq \mathcal{I}$ is then determined as the sum over $\mathcal{I}'$ of the scores per instance:

- The *Mean Standard Score of algorithm a*: $\mathrm{MSS}_a = \frac{1}{|\mathcal{I}'|} \sum_{i \in \mathcal{I}'} \mathrm{MSS}_a(i)$
- The *Normalized Score of algorithm a*: $\mathrm{NS}_a = \frac{1}{|\mathcal{I}'|} \sum_{i \in \mathcal{I}'} \mathrm{NS}_a(i)$
- The *Model Checking Contest score of a*: $\mathrm{MCC}_a = \frac{1}{|\mathcal{I}'|} \sum_{i \in \mathcal{I}'} \mathrm{MCC}_a(i)$

MSS requires a certain amount of samples to be significant. Therefore, we apply it only for those model instances were all our tested algorithms terminated in the time bound. For those instances where not all algorithm terminated, we apply NS. We decided to test the MCC score to check if it is a good or a biased metric, when compared to the standard score.

[1] We actually use a simplified version where answer correctness is not considered.

Table 1. Performance of the ordering algorithms using the MCC2016 models.

		Num. instances				Average scores			
Algorithm a	$\mathcal{A}$	applied	solved	optimal	uniq.	NS_a	MSS^*_a	NS^*_a	MCC_a
`FORCE-PTS`	$\mathcal{A}_{\mathrm{Gen}}$	386	259	23	1	0.699	-0.071	0.552	33.63
`FORCE-NES`	$\mathcal{A}_{\mathrm{Gen}}$	386	267	27	0	0.657	**-0.275**	0.505	34.88
`FORCE-WES1`	$\mathcal{A}_{\mathrm{Gen}}$	386	263	23	0	0.663	-0.228	0.505	34.13
`CM`	$\mathcal{A}_{\mathrm{Gen}}$	386	290	69	2	0.586	-0.076	0.449	40.35
`KING`	$\mathcal{A}_{\mathrm{Gen}}$	386	284	36	2	0.635	-0.032	0.504	37.55
`SLO`	$\mathcal{A}_{\mathrm{Gen}}$	386	**340**	71	**5**	0.534	-0.125	0.471	**46.69**
`NOACK`	$\mathcal{A}_{\mathrm{Gen}}$	386	312	51	0	0.535	-0.052	**0.425**	41.96
`TOV`	$\mathcal{A}_{\mathrm{Gen}}$	386	325	**75**	2	**0.524**	-0.043	0.435	45.07
`MCL`	$\mathcal{A}_{\mathrm{Gen}}$	386	250	27	4	0.767	0.676	0.640	32.76
`FORCE-P`	$\mathcal{A}_{\mathrm{PSF}}$	328	230	33	0	0.604	-0.292	0.435	36.07
`P`	$\mathcal{A}_{\mathrm{PSF}}$	328	258	26	3	0.729	0.623	0.656	39.00
`FORCE-NU`	$\mathcal{A}_{\mathrm{NU}}$	65	38	11	1	0.711	-0.364	0.506	31.01
`NOACK-NU`	$\mathcal{A}_{\mathrm{NU}}$	65	47	3	0	0.777	0.049	0.692	34.70
`TOV-NU`	$\mathcal{A}_{\mathrm{NU}}$	65	45	1	0	0.773	0.122	0.672	32.86

Table 1 describes the general summary of the benchmark results. For each algorithm, we report again its requirement class ($\mathcal{A}_{\mathrm{Gen}}, \mathcal{A}_{\mathrm{PSF}}, \mathcal{A}_{\mathrm{NU}}$). The table reports the number of instances where: the algorithm is applied, the algorithm finishes in the time and memory bounds, the number of times the algorithm finds the best ordering among the others, and the number of times the algorithm is the only one capable of solving an instance. The last four columns report: the NS score on all instances NS_a; the MSS score on completed instances (MSS^*_a); the NS score on completed instances (NS^*_a); and the MCC_a score per instance. Remember that for NS and MSS, a lower score is better, while for MCC a higher score is better.

From the table, it emerges that `TOV` and `SLO` are the methods that perform better, with a clear margin. Other methods, like `FORCE-P` and `FORCE-NU` have the worst average behaviour, even if they perform well when they are capable of solving an instance. Considering the column of "optimal" instances, some methods seem to perform well, like `CM`, but as we shall see later in this section, this is a bias caused by the uneven number of instances per model (i.e. some models have more instances than others). To our surprise, both `MCL` and `P` methods show only a mediocre average performance even if there is a certain number of instances where they perform particularly well.

In general, most instances are solved by more than one algorithm. In the rest of the section we go into model details, first considering the performance of the algorithms, and then by ranking the methods considering either instances($\mathcal{I}$, $\mathcal{I}_{\mathrm{PSF}}$, $\mathcal{I}_{\mathrm{NU}}$) or models($\mathcal{M}$, $\mathcal{M}_{\mathrm{PSF}}$, $\mathcal{M}_{\mathrm{NU}}$).

Efficiency of variable ordering algorithms. Table 2 reports the times required to generate the variable ordering. The table reveals several problems and inef-

Table 2. Time distribution for the tested variable ordering methods, ordered by $\mathbb{E}[t]$.

Method Name	$t<0.01$	$0.01<t<1$	$1<t<10$	$t>10$	d.n.f.	$\mathbb{E}[t]$	$\sigma[t]$
KING	559	44	1	0	0	0.006	0.056
CM	558	45	1	0	0	0.006	0.059
MCL	522	80	2	0	22	0.016	0.129
FORCE-NES	371	220	13	0	4	0.090	0.423
FORCE-WES1	367	222	15	0	4	0.095	0.432
FORCE-PTS	368	221	15	0	4	0.098	0.442
SLO	510	87	5	2	0	0.211	3.602
FORCE-NU	409	131	38	26	0	0.708	2.235
TOV	434	142	21	7	0	0.854	9.092
NOACK	435	142	20	7	0	0.869	9.200
NOACK-NU	433	144	19	8	0	0.879	9.505
TOV-NU	430	146	20	8	0	0.927	9.849
FORCE-P	217	281	78	28	74	1.016	2.474
P	510	54	14	26	103	14.621	159.891

ficiencies of some of these methods in our implementation. In particular, our implementation of the Noack and Tovchigrechko methods are simple prototypes, and have at least the cost of $O(|V|^2)$. The P algorithm also shows a severe performance problem, with a cost that does not match its theoretical complexity. Surprisingly, the cost of Sloan is quite high. The library documentation states that its cost is $O(\log(m) \cdot |E|)$ with $m = \max\{\mathit{degree}(v) \mid v \in V\}$, but the numerical analysis seems to suggest that the library implementation could have some hidden inefficiency.

3.1 Results of the benchmark

Figure 2 shows the benchmark results, separated by instance class and algorithm class. The plots on the left (1, 3, and 5) report the results on the instances that are solved by all algorithms in the given time and memory limits ("Easy instances" hereafter), while those on the right report the results for all instances solved by at least one algorithm ("All instances" hereafter). In the left plots we report the three tested metrics, while on the right plots we discard the MSS, since the available samples for each instance may vary and could be too low for the Gaussian assumption. To fit all scores in a single plot we have rescaled the score values in the $[0, 1]$ range.

Algorithms are sorted by their NS score, best one on the left. The top row (plot 1 and 2) considers the $\mathcal{A}_{\text{Gen}}$ methods on all $\mathcal{I}$ instances. The center row considers the $\mathcal{A}_{\text{Gen}} \cup \mathcal{A}_{\text{PSF}}$ methods on the $\mathcal{I}_{\text{PSF}}$ instances. The bottom row considers the $\mathcal{A}_{\text{Gen}} \cup \mathcal{A}_{\text{NU}}$ methods on the $\mathcal{I}_{\text{NU}}$ instances. Plots 1, 3, and 5 consider 187, 145 and 11 instances, respectively, which are the "easy" instances.

All three scores seem to provide (almost) consistent rankings. The only exception is plot 5. We suspect that this discrepancy can be explained by the small number of available instances (i.e. 11). The MCC metric is not very significant

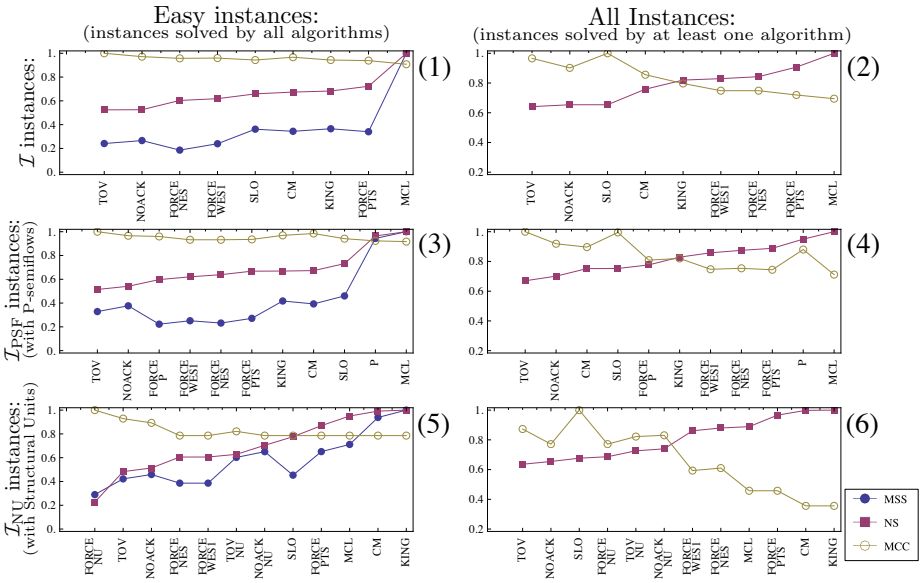


Fig. 2. Benchmark results, weighted by instance.

when we consider only the subset of instances solved by all algorithms, since it does not reward those methods that complete more often that the others. It is instead more significant for the right plots, which consider also the instances where algorithms do not finish.

In general, we can observe that FORCE methods seem to provide better variable orderings when the easy instances are considered (left plots). When we instead consider all the instances, FORCE methods appear to be less efficient. This suggests that FORCE methods could not scale well on larger instances.

We observe a marginally favorable behaviour of the two methods `FORCE-P` and `FORCE-NU` for the models with P-semiflows and structural units, compared to the standard FORCE method. The general trend that seems to emerge from these plots is that `TOV`, `NOACK` and `SLO` have the best average performance. This is particularly evident in plot 2, which refers to the behaviour on "all" instances.

One problem of this benchmark setup is that the MCC model set is made by multiple parametric instances of the same model, and the number of model instances vary largely (some models have just one, others have up to 40 instances). Usually, the relative performances of each algorithm on different instances of the same model are similar. Thus, an instance-oriented benchmark is biased toward those models that have more instances. Therefore, we consider a modified version of the three metrics MSS, NS and MCC, where the value is normalized against the number of instances $\mathcal{I}_m$ of each considered model $m \in \mathcal{M}'$. Therefore, MSS_a is redefined as: $\text{MSS}_a = \frac{1}{|\mathcal{M}'|} \sum_{m \in \mathcal{M}'} \frac{1}{|\mathcal{I}_m|} \sum_{i \in \mathcal{I}_m} \text{MSS}_a(i)$, and NS and MCC are modified analogously.

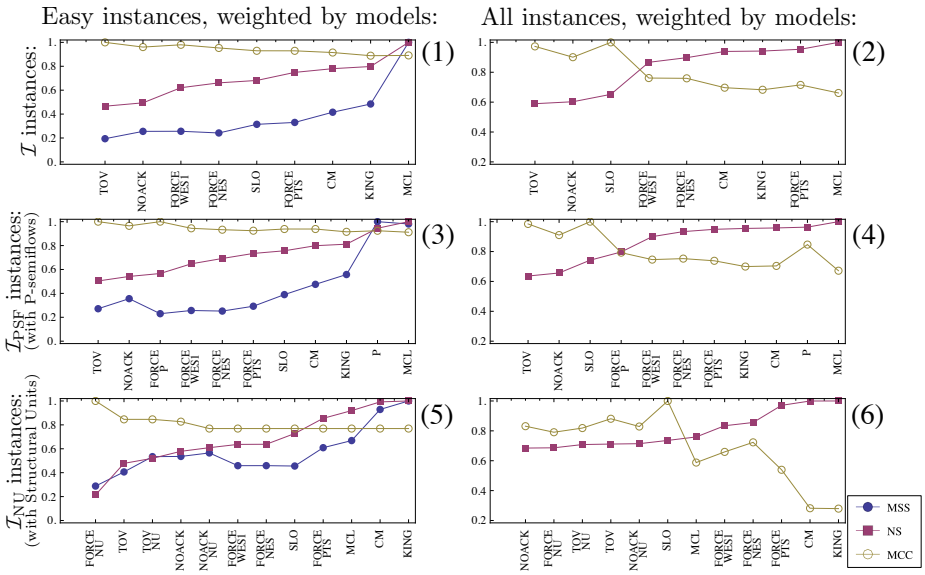


Fig. 3. Benchmark results, weighted by model.

Figure 3 shows the benchmark results weighted by models. Plots 1, 3, and 5 consider 48, 42 and 5 models, respectively, which are those models which have at least an "easy" instance. The main difference observed is the performance of the Cuthill-Mckee method, which on model-average does not seem to perform well. This is explained by the fact that `CM` performs well on three models (BridgeAndVehicle, Diffusion2D and SmallOperatingSystem) that have a large number of instances. But when we look from a model point-of-view, the performance of `CM` drops down.

As we have already stated before, both `MCL` and `P` methods are consistently the worst methods. In addition, neither `TOV-NU` nor `NOACK-NU` do not seem to take advantage of the prioritization on structural units, when compared to their base counterparts.

We may also observe that while `TOV` has the highest NS and MSS scores, `SLO` has the highest MCC score. To investigate this behaviour we look at the score distributions of the algorithms in Figure 4.

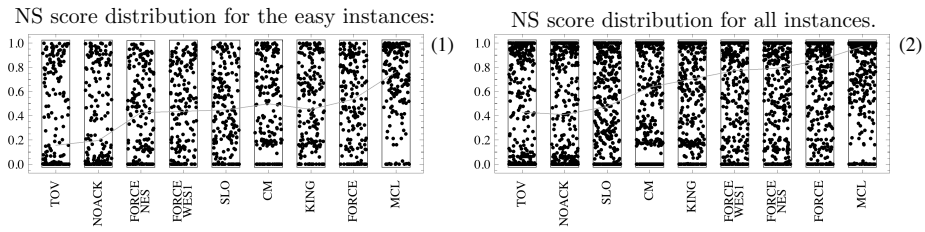


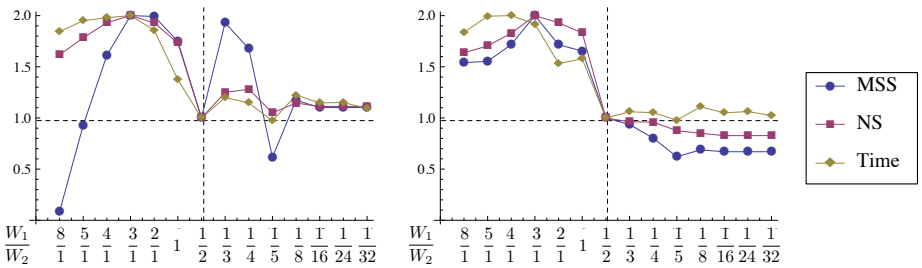
Fig. 4. Score distributions for the by Instance case.

Plot 1 and 2 of Figure 4 show the NS score distributions on the "easier" and "all" instances. The y coordinate of each dot represents the score obtained by an algorithm for an instance. Each box width is chosen large enough to observe the point density. In both plots `TOV` and `NOACK` have a behaviour that is more polarized than `SLO`, which means that usually either they produce an almost-optimal variable ordering, or they have a higher chance of failing at producing one. On the other hand, `SLO` seems to have a more balanced behaviour on most instances, as highlighted by the MCC score, even if it provides the best variable ordering fewer times than `TOV`. This happens because the MCC score is designed to give almost identical points to all algorithms that complete (except for the best algorithm, which takes extra points), regardless of the "quality" of the variable ordering. Therefore, the method that *on-average* performs better will have a higher MCC score.

4 Parameter Estimation of the Sloan priority function weights

Since Sloan seems to provide very good variable orders on average, we have investigated whether the parametric priority function $P(v')$ can be tuned, since the implementation of Sloan available in the library has been developed for a different purpose than ordering MDD variables.

Figure 5 shows the MSS, NS and Time lines of the Sloan algorithm applied to 271 instances. The results have been obtained considering the subset of $\mathcal{I}$ that terminate in less than 10 minutes. These instances belong to 56 different models. The lines are rescaled, and centered around the $W_1 = 1, W_2 = 2$ point, which are the standard coefficients of the priority function.



(**A**) Scores per instances. (**B**) Scores per models.

Fig. 5. Performance trends of Sloan method for different weight values.

We experiment a parameter space of the $\frac{W_1}{W_2}$ ratio in the range going from $\frac{8}{1}$ to $\frac{1}{32}$. The figure shows that the ratio of $\frac{1}{2}$, which is the standard value for the algorithm as implemented in the libraries, is not necessarily the best choice on average. As before, the per-instance plots shows slightly different results than

the per-model plot. The per-model plot shows that, on average, a higher value of W_2 is beneficial to find a better variable ordering.

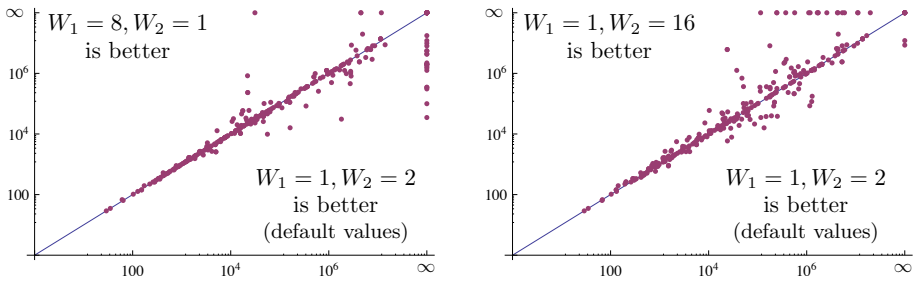


Fig. 6. Comparison of 3 variations of the weights in the Sloan priority function $P(v')$.

We now consider the $\frac{8}{1}$, $\frac{1}{2}$ and $\frac{1}{16}$ versions for a more detailed analysis on the full set of 386 instances with a time limit of 75 minutes, depicted in Figure 6. Each point represents an instance, where its x and y coordinates are the MDD peak values computed using the compared parameter variations. Coordinates are represented on a log-log scale. Instances that did not finish are considered as having an infinite peak node value, and are represented on the top (resp. right) sides of the plot. Most instances stay close to the diagonal, which means that there is almost no difference between the two parametric variations on that instance. Some instance instead show different MDD peaks, and are mainly concentrated toward the variations with $W_2 > W_1$. We shall now refer to the parametric variation of Sloan with $W_1 = 1, W_2 = 16$ as `SLO:1/16`.

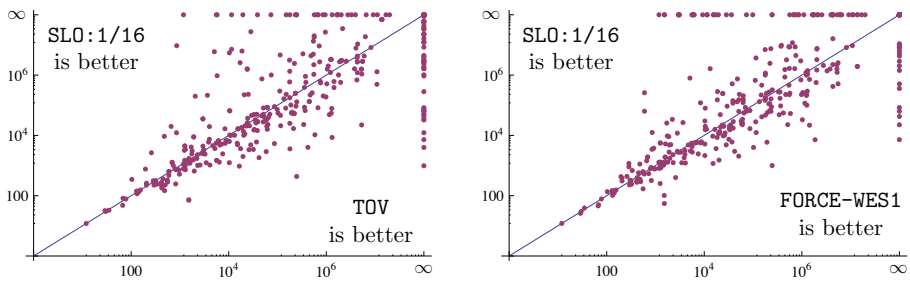


Fig. 7. Comparison of Sloan against Tovchigrechko and FORCE-WES1 methods.

Figure 7 shows the comparison of `SLO:1/16` against `TOV` and `FORCE-WES1`. Unlike the previous case, the plots show that many instances are far from the diagonal, which means that the three algorithms perform well on different instances.

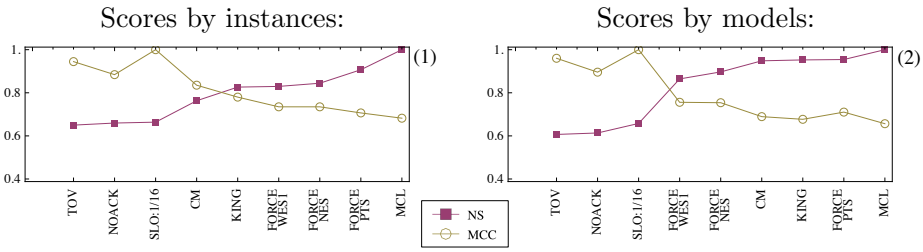


Fig. 8. Benchmark results after the addition of the optimized Sloan method.

Finally, Figure 8 shows the benchmark results after the substitution of `SLO` with `SLO:1/16`. The plots report both the NS and MCC metrics by instances (left) and by models (right) on "All" instances. `SLO:1/16` performs better than `SLO` in 71 instances (∼13 models) with a MDD peak reduction of 37% (24% for models), while `SLO` beats `SLO:1/16` in 66 instances (∼18 models) with a peak reduction of 56% (20% for models). For 200 instances (∼28 models) the variable ordering remains unchanged. Of the 386 tested models, `SLO` solves 340 of them, and in 71 cases it provides the best ordering among the other algorithms. Instead, `SLO:1/16` solves 341 instances, being optimal in 81 cases. These data show the favorable performance of `SLO:1/16`. This also confirms the observations of Figure 6, i.e. the set of models where Sloan works well remains almost unaltered, but `SLO:1/16` has a higher chance of finding a better ordering.

5 Conclusions and Future works

In this paper we presented a comparative benchmark of 14 variable ordering algorithms. Some of these algorithms are popular among Petri net based model checkers, while others have been defined to investigate the use of structural information for variable orderings. We observed that the Tovchigrechko/Noack methods and the Sloan method have the best performances, and their effectiveness cover different subsets of model instances. While the methods of Tovchigrechko/Noack were designed for Petri net models, the method of Sloan is a standard algorithm for bandwidth reduction of matrices, whose effectiveness for variable ordering was pointed out only recently in [19] and [22]. We observed that Sloan for variable ordering generation can be improved by changing the priority function weights. The parameter estimation on Sloan seems to suggest that the gradient (controlled by W_2) has a higher impact than keeping a low matrix profile (controlled by W_1). We conjecture that other algorithms, like `TOV`, `FORCE` or `P`, could be improved by using a super-imposed gradient. We did not observe significant improvements for those algorithms that use structural information of the net, and in some cases (`TOV` and `NOACK`) we even observed a degradation in performance. It is therefore unclear whether and how this type of structural information can be used to improve variable orderings. We also tested three different metric scores. We observed an agreement between the MSS and the NS score, which is nice since NS can be used even when few algorithms complete.

We also observed that MCC is a good score, that favours a different aspect than MSS/NS (average behaviour over better ordering). The use of a per-model weight on the scores has helped in identifying a bias in the benchmark results. We think that some form of per-model weight is necessary when using the MCC model set.

It should be noted that we observed a different ranking than the one reported in [22]. That paper deals with metrics for variable ordering (without RS construction), but in the last section the authors report a small experimental assessment similar to our benchmark. In that assessment Tovchigrechko was not tested, and the best algorithms were mostly Sloan and FORCE. For Sloan, they used the default parameter setting with a rather different symmetrization of the adjacency matrix, while it is not clear what variation of FORCE was used. This, together with the fact that the tested model set was different (106 instances), makes it difficult to draw any definite conclusions.

Acknowledgement.

We would like to thank the MCC team and all colleagues that collaborated with them for the construction of the MCC database of models, and the Meddly library developers.

References

1. Ajmone Marsan, M., Conte, G., Balbo, G.: A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. ACM Transactions on Computer Systems 2, 93–122 (May 1984)
2. Aldinucci, M., Bagnasco, S., Lusso, S., Pasteris, P., Vallero, S., Rabellino, S.: The Open Computing Cluster for Advanced data Manipulation (OCCAM). In: 22nd Int. Conf. on Computing in High Energy and Nuclear Physics. San Francisco (2016)
3. Aloul, F.A., Markov, I.L., Sakallah, K.A.: FORCE: A fast and easy-to-implement variable-ordering heuristic. In: Proc. of GLSVLSI. pp. 116–119. ACM, NY (2003)
4. Amparore, E.G.: Reengineering the editor of the greatspn framework. In: PNSE@ Petri Nets. pp. 153–170 (2015)
5. Amparore, E.G., Balbo, G., Beccuti, M., Donatelli, S., Franceschinis, G.: 30 Years of GreatSPN, chap. In: Principles of Performance and Reliability Modeling and Evaluation: Essays in Honor of Kishor Trivedi, pp. 227–254. Springer, Cham (2016)
6. Amparore, E.G., Beccuti, M., Donatelli, S.: (stochastic) model checking in GreatSPN. In: Ciardo, G., Kindler, E. (eds.) 35th Int. Conf. Application and Theory of Petri Nets and Concurrency, Tunis. pp. 354–363. Springer, Cham (2014)
7. Baarir, S., Beccuti, M., Cerotti, D., Pierro, M.D., Donatelli, S., Franceschinis, G.: The GreatSPN tool: recent enhancements. Performance Eval. 36(4), 4–9 (2009)
8. Babar, J., Miner, A.: Meddly: Multi-terminal and edge-valued decision diagram library. In: Quantitative Evaluation of Systems, International Conference on. pp. 195–196. IEEE Computer Society, Los Alamitos, CA, USA (2010)
9. Bollig, B., Wegener, I.: Improving the variable ordering of OBDDs is NP-complete. IEEE Trans. Comput. 45(9), 993–1002 (Sep 1996)

10. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers 35, 677–691 (August 1986)
11. Burch, J.R., Clarke, E.M., Long, D.E.: Symbolic model checking with partitioned transition relations. In: IFIP TC10/WG 10.5 Very Large Scale Integration. pp. 49–58. North-Holland (1991)
12. Ciardo, G., L uttgen, G., Siminiceanu, R.: Saturation: An efficient iteration strategy for symbolic state-space generation. In: TACAS'01. pp. 328–342 (2001)
13. Ciardo, G., Marmorstein, R., Siminiceanu, R.: Saturation unbound. In: In Proc. of TACAS 2003. pp. 379–393. LNCS 2619, Springer (apr 2003)
14. Ciardo, G., Yu, A.J.: Saturation-based symbolic reachability analysis using conjunctive and disjunctive partitioning. In: Proc. CHARME. pp. 146–161. LNCS 3725, Springer (2005)
15. Cuthill, E., McKee, J.: Reducing the bandwidth of sparse symmetric matrices. In: Proc. of the 1969 24th National Conference. pp. 157–172. ACM, New York (1969)
16. F. Kordon et all: Complete Results for the 2016 Edition of the Model Checking Contest. `http://mcc.lip6.fr/2016/results.php` (June 2016)
17. Garavel, H.: Nested-Unit Petri Nets: A structural means to increase efficiency and scalability of verification on elementary nets. In: 36th Int. Conf. Application and Theory of Petri Nets, Brussels. pp. 179–199. Springer, Cham (2015)
18. Heiner, M., Rohr, C., Schwarick, M., Tovchigrechko, A.A.: MARCIE's secrets of efficient model checking. In: Transactions on Petri Nets and Other Models of Concurrency XI. pp. 286–296. Springer, Heidelberg (2016)
19. Kamp, E.: Bandwidth, profile and wavefront reduction for static variable ordering in symbolic model checking. Tech. rep., University of Twente (June, 2015)
20. King, I.P.: An automatic reordering scheme for simultaneous equations derived from network systems. Journal of Numerical Methods in Eng. 2(4), 523–533 (1970)
21. Kumfert, G., Pothen, A.: Two improved algorithms for envelope and wavefront reduction. BIT Numerical Mathematics 37(3), 559–590 (1997)
22. Meijer, J., van de Pol, J.: Bandwidth and wavefront reduction for static variable ordering in symbolic reachability analysis. In: bth NASA Formal Methods, 2016, Minneapolis. pp. 255–271. Springer, Cham (2016)
23. Miner, A.S.: Implicit GSPN reachability set generation using decision diagrams. Performance Evaluation 56(1-4), 145–165 (mar 2004)
24. Noack, A.: A ZBDD package for efficient model checking of Petri nets (in German). Ph.D. thesis, BTU Cottbus, Department of CS (1999)
25. Rice, M., Kulhari, S.: A survey of static variable ordering heuristics for efficient BDD/MDD construction. Tech. rep., University of California (2008)
26. Siminiceanu, R.I., Ciardo, G.: New metrics for static variable ordering in decision diagrams. In: 12th Int. Conf. TACAS 2006. pp. 90–104. Springer, Heidelberg (2006)
27. Sloan, S.W.: An algorithm for profile and wavefront reduction of sparse matrices. International Journal for Numerical Methods in Engineering 23(2), 239–251 (1986)
28. Tovchigrechko, A.: Model checking using interval decision diagrams. Ph.D. thesis, BTU Cottbus, Department of CS (2008)
29. Van Dongen, S.: A cluster algorithm for graphs. Inform. systems 10, 1–40 (2000)

On the Resource Equivalences in Petri Nets with Invisible Transitions*

Vladimir A. Bashkin

Yaroslavl State University, Yaroslavl, 150000, Russia
v_bashkin@mail.ru

Abstract. Two resources (submarkings) are called similar if in any marking any one of them can be replaced by another one without affecting the net's behaviour (modulo marking bisimulation). It is known that resource similarity is undecidable for general labelled Petri nets. In this paper we study the properties of resource similarity and resource bisimulation (a subset of complete similarity relation, closed under transition firing) in Petri nets with invisible transitions (where some transitions may be unlabelled and hence invisible for external observer). It is shown that for a proper subclass — p-saturated nets — the weak transfer property of resource bisimulation can be effectively checked.

1 Introduction

In this paper the behavior of Petri nets is investigated from the standpoint of bisimulation equivalence. The fundamental notion of bisimulation was introduced by R. Milner [9] and D. Park [10]. Two markings of a Petri net are called bisimilar if the choice of each of them as an initial marking gives the same visible behavior of the net. In [7] P. Jančar proved that bisimulation equivalence of markings is undecidable for a general Petri net.

In [1] C. Autant et al. introduced a notion of place bisimulation — a decidable bisimulation-induced equivalence on the finite set of places, that allows to find out some non-trivial behaviour-preserving net reductions. This relation and its applications were studied in [1, 2, 12].

The notion of resource similarity was introduced in [3]. In general a resource is a submarking. Two resources are similar if, having replaced one resource in any marking by another, we obtain the same observed behavior of the net. Resource bisimulation is a particular case of similarity that is closed under transition firing. Place bisimulation is a proper subset of resource bisimulation. Note that, unlike the place bisimulation [1], resource similarity and bisimulation are defined on the infinite set (of resources/submarkings).

Resource similarity and its modifications where studied in [3–5]. In particular it was proven that resource similarity is undecidable. However, it was shown that resource bisimulation can be effectively approximated and used as a basis of net reductions and adaptive control. For an overview, see [8].

* This work is supported by Russian Fund for Basic Research (project 17-07-00823).

In this paper we consider an important generalization of labelled Petri nets, where some transitions may be unlabelled and hence invisible for external observer. Quite often when analyzing the system there is a need to abstract from the excessive information about its behavior. For example, it is convenient to hide all transitions, corresponding to the internal actions of the system. The information obtained in this case can be useful, in particular, to detect additional properties of the system in terms of its interaction with the environment.

Place bisimulations in Petri nets with invisible transitions were studied by C. Autant et al. in [2]. It was shown that unlabelled sequences of steps significantly complicate the calculations. However, there are specific nontrivial subclasses of Petri nets with invisible transitions, that have some nice properties w.r.t. place bisimulation.

In this paper we basically apply a similar approach to the resource equivalences. It is shown that, despite their non-trivial infinite structure, resource bisimulations can be effectively computed even in the case of nets with invisible transitions. In particular, it is shown that for a proper subclass — p-saturated nets — the weak transfer property of resource τp-bisimulation can be effectively checked. Moreover, we can underapproximate the largest τp-bisimulation by a parameterized algorithm.

The paper is organized as follows. Section 2 contains basic definitions. Specifically, in Subsection 2.1 we give some technical notions and lemmata on the properties of additively-transitively closed relations on multisets. Subsection 2.2 contains definitions of Petri nets and bisimulations. Subsections 2.3 and 2.4 give a short review on Petri net resources and resource equivalences (similarity and bisimulation). Section 3 deals with invisible transitions. In Subsections 3.1 and 3.2 we define the straightforward τ-generalizations of resource equivalences and study their properties. Subsections 3.3 and 3.4 describe the subclass of p-saturated nets and the corresponding notion of τp-bisimulation. In Subsection 3.5 we present an algorithm, computing the parameterized underapproximation of largest τp-bisimulation. Section 4 contains some conclusions.

2 Preliminaries

2.1 Relations on multisets

Let X be a finite set. A *multiset* m over a set X is a mapping $m : X \to Nat$, where Nat is the set of natural numbers (including zero), i.e. a multiset may contain several copies of the same element.

For two multisets m, m' we write $m \subseteq m'$ iff $\forall x \in X : m(x) \leq m'(x)$ (the inclusion relation). The sum and the union of two multisets m and m' are defined as usual: $\forall x \in X : m + m'(x) = m(x) + m'(x),\ m \cup m'(x) = max(m(x), m'(x))$. By $\mathcal{M}(X)$ we denote the set of all finite multisets over X.

Non-negative integer vectors are often used to encode multisets. Actually, the set of all multisets over finite X is a homomorphic image of $Nat^{|X|}$.

A binary relation $R \subseteq Nat^k \times Nat^k$ is a congruence if it is an equivalence relation and whenever $(v, w) \in R$ then $(v + u, w + u) \in R$ (here '+' denotes coordinate-wise addition). It was proved by L. Redei [11] that every congruence on Nat^k is generated by a finite set of pairs. Later P. Jančar [7] and J. Hirshfeld [6] presented a shorter proof and also showed that every congruence on Nat^k is a semilinear relation, i.e. it is a finite union of linear sets.

Let R^{AT} denote the additive-transitive closure (AT-closure) of the relation $R \subseteq \mathcal{M}(X) \times \mathcal{M}(X)$ (the minimal congruence, containing R).

Let $B \subseteq \mathcal{M}(X) \times \mathcal{M}(X)$ be a binary relation on multisets. A relation B' is called an *AT-basis* of B iff $(B')^{AT} = B^{AT}$. An AT-basis B' is called *minimal* iff there is no $B'' \subset B'$ such that $(B'')^{AT} = B^{AT}$.

Now we construct a special kind of minimal AT-basis for B. Define a partial order $\sqsubseteq$ on the set $B \subseteq \mathcal{M}(X) \times \mathcal{M}(X)$ of pairs of multisets as follows:

1. For loop (i.e. reflexive) pairs let

$$(r_1, r_1) \sqsubseteq (r_2, r_2) \overset{def}{\Leftrightarrow} r_1 \subseteq r_2;$$

2. For two non-loop pairs, the maximal loop constituents and the addend pairs of nonintersecting multisets are compared separately

$$(r_1 + o_1, r_1 + o_1') \sqsubseteq (r_2 + o_2, r_2 + o_2') \overset{def}{\Leftrightarrow}$$

$$\overset{def}{\Leftrightarrow} o_1 \cap o_1' = \emptyset \ \& \ o_2 \cap o_2' = \emptyset \ \& \ r_1 \subseteq r_2 \ \& \ o_1 \subseteq o_2 \ \& \ o_1' \subseteq o_2'.$$

3. A loop pair and a non-loop pair are always incomparable.

Let B_s denote the set of all minimal (with respect to $\sqsubseteq$) elements of B^{AT}.

Theorem 1. *[4] Let $B \subseteq \mathcal{M}(X) \times \mathcal{M}(X)$ be a symmetric and reflexive relation. Then B_s is an AT-basis of B and B_s is finite.*

We call B_s the *ground basis* of B. Obviously, it is finite.

There is also a useful

Lemma 1. *[4] Let $B \subseteq \mathcal{M}(X) \times \mathcal{M}(X)$ be a symmetric and reflexive relation, $(r, s) \in B^{AT}$. Then there exists a finite chain of pairs*

$$(r, a_1), (a_1, a_2), \ldots, (a_{k-1}, a_k), (a_k, s) \in (B_s)^A,$$

where $(B_s)^A$ is the additive closure of B_s.

2.2 Labelled Petri nets and bisimulations

Let P and T be disjoint sets of *places* and *transitions* and let $F : (P \times T) \cup (T \times P) \to Nat$. Then $N = (P, T, F)$ is a *Petri net*. A *marking* in a Petri net is a function $M : P \to Nat$, mapping each place to some natural number (possibly zero). Thus a marking may be considered as a multiset over the set of places.

Pictorially, P-elements are represented by circles, T-elements by boxes, and the flow relation F by directed arcs. Places may carry tokens represented by filled circles. A current marking M is designated by putting $M(p)$ tokens into each place $p \in P$. Tokens residing in a place are often interpreted as resources of some type consumed or produced by a transition firing. A marked Petri net (N, M_0) is a Petri net N together with an initial marking M_0.

For a transition $t \in T$ the *preset* ${}^\bullet t$ and the *postset* $t^\bullet$ are defined as the multisets over P such that ${}^\bullet t(p) = F(p,t)$ and $t^\bullet(p) = F(t,p)$ for each $p \in P$. A transition $t \in T$ is *enabled* in a marking M iff $\forall p \in P\ M(p) \geq F(p,t)$. An enabled transition t may *fire* yielding a new marking $M' =_{\text{def}} M - {}^\bullet t + t^\bullet$, i.e. $M'(p) = M(p) - F(p,t) + F(t,p)$ for each $p \in P$ (denoted $M \xrightarrow{t} M'$).

The transitions may *fire in parallel* (concurrently), if there are enough tokens for all of them. In particular, the transition may fire in parallel with itself. The concurrent firing of a multiset of transitions is called *a parallel step.* The precondition and postcondition for a multiset of transitions $\alpha \in \mathcal{M}(T)$ are:

$$ {}^\bullet\alpha =_{def} \sum_{t\in\alpha} {}^\bullet t, \qquad \alpha^\bullet =_{def} \sum_{t\in\alpha} t^\bullet. $$

Obviously, ${}^\bullet(\alpha+\beta) = {}^\bullet\alpha + {}^\bullet\beta, (\alpha+\beta)^\bullet = \alpha^\bullet + \beta^\bullet$.

To observe a net behavior transitions are labelled by special labels representing observable actions or events. Let Act be a set of action names. A *labelled Petri net* is a tuple $N = (P, T, F, l)$, where (P, T, F) is a Petri net and $l : T \to Act$ is a labelling function. It can be generalized to sequences:

for $\alpha \in T^*$ s.t. $\alpha = t\beta$ with $t \in T$ and $\beta \in T^*$ we have $l(\alpha) =_{def} l(t)l(\beta)$.

And also to multisets of transitions:

$$ \text{for } \alpha \in \mathcal{M}(T) \qquad l(\alpha) =_{def} \textstyle\sum_{t\in\alpha} l(t). $$

Here we use not a union but a sum of multisets.

Let $N = (P, T, F, l)$ be a labelled Petri net. We say that a relation $R \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ conforms to the *transfer property* iff for all $(M_1, M_2) \in R$ and for every step $t \in T$, s.t. $M_1 \xrightarrow{t} M_1'$, there exists an imitating step $u \in T$, s.t. $l(t) = l(u)$, $M_2 \xrightarrow{u} M_2'$ and $(M_1', M_2') \in R$.

A relation R is called a *marking bisimulation*, if both R and R^{-1} conform to the transfer property.

For every labelled Petri net there exists the largest marking bisimulation (denoted by $\sim$) and this bisimulation is an equivalence. It was proved by P. Jančar [7], that the marking bisimulation is undecidable for Petri nets.

2.3 Resource similarity

Informally, resources are parts of markings which may or may not provide this or that kind of net behavior.

Definition 1. *[4] Let $N = (P, T, W, l)$ be a labelled Petri net. A* resource $R \in \mathcal{M}(P)$ *in a Petri net N is a multiset over the set of places P.*

Resources r and s in N are called similar *(denoted $r \approx s$) iff for every marking $R \in \mathcal{M}(P)$, $r \subseteq R$ implies $R \sim R - r + s$.*

Thus if two resources are similar, then in every marking each of these resources can be replaced by another without changing the observable system's behavior. Some examples of similar resources are shown in Fig. 1.

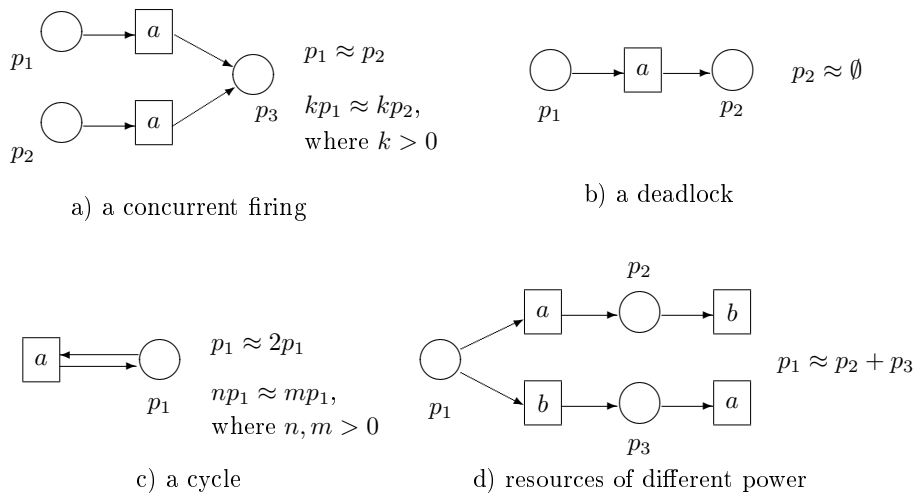


Fig. 1. Examples of similar resources.

Figure a) shows a Petri net containing two transitions labeled with the same label a and leading to the same marking p_3. Here the resources p_1 and p_2 are similar, as they lead to a completely identical observable behavior — action a producing a single token in p_3. Moreover, all the resources containing the same number of tokens in p_1 and p_2 are similar.

Figure b) shows a simple net consisting of a single transition. In this case the resource p_2 is similar to an empty resource, since it does not affect the behavior of the net (the place p_2 is redundant).

Figure c) depicts a cycle consisting of one transition and one place. Note that the set of markings of this net can be divided into two disjoint subsets — empty marking and all the others. With empty marking, the transition can not fire, for all others — it can fire any number of times. Note that for this net the largest marking bisimulation and the resource similarity coincide.

Figure d) shows a more complex network. We have $p_1 \approx p_2 + p_3$, that is, replacing one token in p_1 by two tokens (one in p_2 and one in p_3) does not affect the observable behavior of the net as a whole.

The similarity relation is an equivalence [4]. Moreover, it is monotonous:

Proposition 1. *[4] Let $N = (P, T, W, l)$ be a labelled Petri net, let r, s, u, v be resources of the net N. Then $r \approx s$ & $u \approx v \;\Rightarrow\; r + u \approx s + v$.*

Hence it has a finite ground basis. Unfortunately, from the undecidability of a stronger relation of place fusion [12] we get

Theorem 2. *[4] The resource similarity is undecidable for labelled Petri nets.*

2.4 Resource bisimulation

We defined a stronger equivalence relation, retaining the observable system's behavior:

Definition 2. *[4] An equivalence relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ is called a resource bisimulation if B^{AT} is a marking bisimulation.*

Note that an AT-closure of a resource similarity is not necessarily a marking bisimulation. The next theorem states some important properties of resource bisimulations.

Theorem 3. *[4] Let N be a labelled Petri net. Then*

1. *if $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ is a resource bisimulation and $(r_1, r_2) \in B$ then $r_1 \approx r_2$;*
2. *if B_1, B_2 are resource bisimulations for N then $B_1 \cup B_2$ is a resource bisimulation for N;*
3. *for any N there exists the largest resource bisimulation (denoted by $B(N)$), and it is an equivalence.*

Therefore $B(N)$ (as well as any other resource bisimulation) also has a finite ground basis.

The AT-closure of a resource bisimulation is a marking bisimulation, and hence, it conforms to the transfer property. Resource bisimulations satisfy a weak variant of the transfer property, when only 'adjacent' markings are considered for a transition t:

We say that a relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ conforms to *the weak transfer property* if for all $(r, s) \in B$, for each $t \in T$, such that ${}^\bullet t \cap r \neq \emptyset$, there exists an imitating transition $u \in T$, such that $l(t) = l(u)$ and, writing M_1 for ${}^\bullet t \cup r$ and M_2 for ${}^\bullet t - r + s$, we have $M_1 \xrightarrow{t} M_1{}'$ and $M_2 \xrightarrow{u} M_2{}'$ with $(M_1', M_2') \in B^{AT}$.

Theorem 4. *[4] A relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ is a resource bisimulation iff B is an equivalence relation and it conforms to the weak transfer property.*

Due to this theorem to check whether a given finite relation B is a resource bisimulation, one needs to verify the weak transfer property for only a finite number of pairs of resources. In [4] we have shown that the largest resource bisimulation for resources with a bounded number of tokens can be effectively constructed (more precisely, it requires $\mathrm{O}(\max\{|P|\,\mathcal{R}^9, |T|^2|P|\,\mathcal{R}^7\})$ steps, where $\mathcal{R}$ is the number of resources in the consideration).

3 Petri nets with invisible transitions

In this section we investigate the possibilities of effectively constructing bisimulation-preserving relations for an extended class of systems — Petri nets with invisible transitions.

To distinguish visible and invisible transitions, a special τ symbol is added to the set of labels: $Act_\tau = Act \cup \{\tau\}$.

Definition 3. *A* labelled Petri net with invisible transitions *is a tuple* $N = (P, T, F, l)$*, where* (P, T, F) *is a Petri net and* $l : T \to Act_\tau$ *is an extended labelling function.*

Let $\sigma, \sigma' \in (Act_\tau)^*$ be sequences of action labels (with τ-s). Denote $\sigma =_\tau \sigma' \Leftrightarrow_{def} \sigma_{|Act} = \sigma'_{|Act}$ ("equal modulo τ"). For example, "$\tau\tau a\tau$" $=_\tau$ "a".

3.1 τ-bisimulation

Let $N = (P, T, F, l)$ be a labelled Petri net with invisible transitions. We say that a relation $R \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ conforms to the *τ-transfer property* iff for all $(M_1, M_2) \in R$ and for every step $t \in T$, s.t. $M_1 \xrightarrow{t} M_1'$, there exists an imitating sequence of steps $\sigma \in T^*$ s.t. $l(t) =_\tau l(\sigma)$, $M_2 \xrightarrow{\sigma} M_2'$ and $(M_1', M_2') \in R$.

A relation R is called a *marking τ-bisimulation*, if both R and R^{-1} conform to the τ-transfer property. The largest τ-bisimulation is denoted by $\sim_\tau$.

Marking bisimulation is a special case of τ-bisimulation (for nets with no τ-s). It is a stronger relation. Consider as an example the net at Fig. 2. Markings p_1 and p_2 are not bisimilar, because at p_2 no transition with label a is active. But they are τ-bisimilar, because the invisible firing of t_2 changes the marking from p_2 to p_1.

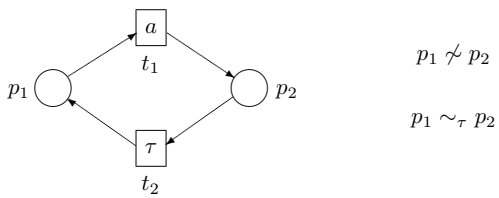


Fig. 2. τ-bisimulation is weaker than bisimulation

In particular, this implies the undecidability of τ-bisimulation in Petri nets with invisible transitions [7].

3.2 Resource similarity and bisimulation

The definition of resource similarity can be naturally generalized to the case of nets with invisible transitions:

Definition 4. *Let $N = (P, T, F, l)$ be a labelled Petri net with invisible transitions. Resources r and s are called* τ-similar *(denoted $r \approx_\tau s$) iff for every marking R, $r \subseteq R$ implies $R \sim_\tau R - r + s$.*

We can show that resource τ-similarity has all basic properties of resource similarity:

Proposition 2. *1. Resource τ-similarity is closed under addition and transitivity; hence it has finite AT-basis.*
2. Resource τ-similarity is undecidable.

Proof. 1) From the definitions.
2) From Th. 2 (note that τ-similarity is a generalization of basic resource similarity).

The definition of resource bisimulation also can be easily generalized:

Definition 5. *Let $N = (P, T, F, l)$ be a labelled Petri net with invisible transitions. An equivalence relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ is called a* resource τ-bisimulation *if B^{AT} is a marking τ-bisimulation.*

Proposition 3. *Let $N = (P, T, F, l)$ be a labelled Petri net with invisible transitions. Then*

1. *if $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ is a resource τ-bisimulation and $(r_1, r_2) \in B$ then $r_1 \approx_\tau r_2$;*
2. *if $B_1, B_2 \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ are resource τ-bisimulations then $B_1 \cup B_2$ is a resource τ-bisimulation;*
3. *for any N there exists the largest resource τ-bisimulation (denoted by $B_\tau(N)$), and it is an equivalence.*

Proof. The first statement follows directly from the definitions. Note, that there exists a resource τ-similarity which is not a τ-bisimulation.

The proof of the second statement is rather long and contains some technical details. It uses the decomposition of a given pair into a transitive chain of pairs, where pairs are constructed as sums of pairs from $(B_1)^{AT}$ and $(B_2)^{AT}$.

The third statement is an immediate corollary of the second. The largest resource τ-bisimulation is the union of all resource τ-bisimulations for N.

Definition 6. *We say that a relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ conforms to the* weak τ-transfer property *if for all $(r, s) \in B$, $t \in T$ s.t. ${}^\bullet t \cap r \neq \emptyset$, there exists an imitating sequence of transitions $\sigma \in T^*$ s.t. $l(t) =_\tau l(\sigma)$ and, denoting $M_1 = {}^\bullet t \cup r$ and $M_2 = {}^\bullet t - r + s$, we have $M_1 \xrightarrow{t} M_1{}'$ and $M_2 \xrightarrow{\sigma} M_2{}'$ with $(M_1', M_2') \in B^{AT}$.*

Th. 4 in the case of Petri nets with invisible transitions works only in one direction:

Proposition 4. *If the relation conforms to the τ-transfer property then it conforms to the weak τ-transfer property; there exist relations, conforming to the weak τ-transfer property and not conforming to the τ-transfer property.*

Proof. ($\Rightarrow$) Since the weak τ-transfer property is the τ-transfer property for a bounded (finite) subset of pairs of resources.

($\not\Leftarrow$) Consider a net at Fig. 3 (this example is taken from [2]) and a relation

$$B = Id(P) \cup \{(p_1, p_2), (p_2, p_1), (p_3, p_4), (p_4, p_3)\}.$$

B conforms to the weak τ-transfer property. At the same time B is not a resource τ-bisimulation. Consider markings $M_1 = p_1 + p_3$ and $M_2 = p_2 + p_4$. The pair (M_1, M_2) belongs to the relation B^{AT}, but the markings are not bisimilar, because an action a is possible at M_2 (transition t_3) and is impossible at M_1.

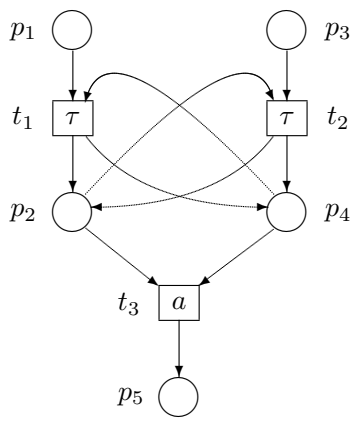


Fig. 3. Th. 4 does not hold for Petri nets with invisible transitions.

Hence the weak τ-transfer property can not be used to construct bisimulation. In the case of systems with invisible transitions it is even more important to strengthen the considered relations and/or to restrict the considered class of Petri nets.

3.3 Saturated nets

There exists a wide and important subclass of Petri nets with invisible transitions for which resource τ-bisimulation can be constructed using weak transfer property — so-called "p-saturated nets". In p-saturated nets [2] the firing of any sequence of transitions with at most one visible label can be simulated by a simultaneous (independent) firing of a certain set of transitions with the same label (called "parallel step").

Denote the set of transition sequences with at most one visible label:

$$T^{\times} =_{def} \{\sigma \in T^* \mid l(\sigma) \in Act_\tau\}.$$

Definition 7. *A labelled Petri net with invisible transitions* $N = (P, T, F, l)$ *is called* p-saturated *(or simply* saturated*), if for any sequence of transitions* $\sigma \in T^{\times}$ *there exists a parallel step* $U \in \mathcal{M}(T)$ *s.t.* $l(U) =_\tau l(\sigma)$, ${}^\bullet U = {}^\bullet\sigma$ *and* $U^\bullet = \sigma^\bullet$.

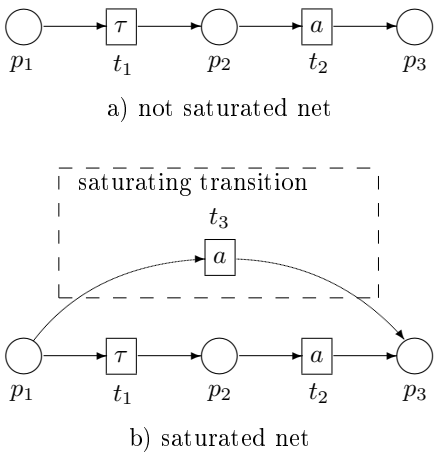


Fig. 4. An example of net saturation

In addition to saturated nets, there is an even broader class of *saturable* Petri nets. These are nets that can be transformed into saturated by adding a finite number of transitions while preserving the behavior of the net (in the sense of τ-bisimularity). On Fig. 4 a saturated net is shown, obtained by adding the transition t_3 to the unsaturated net.

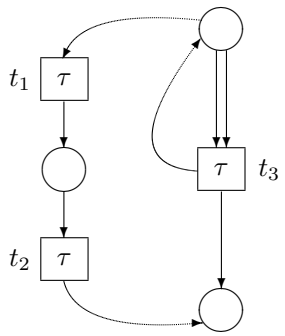
It is known [2] that a net is p-saturated iff it is $2p$-saturated, i.e. all sequences of length 2 are saturated by parallel steps.

Not all nets are saturable [2]. An example is given at Fig. 5.

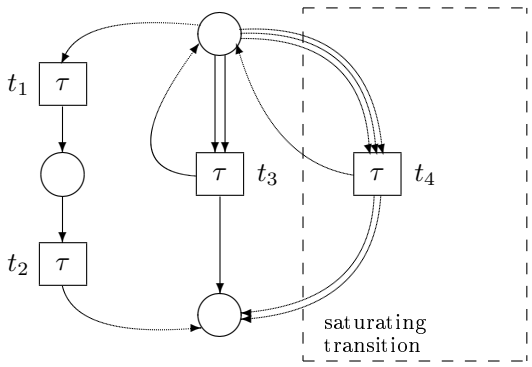
It is also easy to see that the net is saturable iff its "invisible subnet" is saturable (i.e. a net, obtained by removing all visible transitions).

3.4 τp-bisimulation

In [2] an equivalence stronger than τ-bisimulation was defined, called τp-bisimulation of markings. The transition in this case is modeled not by a sequence of transitions, but by a parallel step.



a) not saturated net



b) first step of "saturation"

Fig. 5. Not saturable net

Definition 8. *[2] Let $N = (P, T, F, l)$ be a labelled Petri net with invisible transitions. We say that a relation $R \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ conforms to the τp-*transfer property *if for all $(M_1, M_2) \in R$ and for each $t \in T$ s.t. $M_1 \xrightarrow{t} M_1'$, there exists an imitating parallel step $U \in \mathcal{M}(T)$ s.t. $l(t) =_\tau l(U)$, $M_2 \xrightarrow{U} M_2'$ and $(M_1', M_2') \in R$.*

Definition 9. *[2] A relation R is called a* marking τp-bisimulation*, if both R and R^{-1} conform to the τp-transfer property.*

It is known [2] that for any net there exists the largest τp-bisimulation (denoted by $\sim_{\tau p}$).

In saturated Petri nets τp-bisimulation coincides with τ-bisimulation [2]:

$$M_1 \sim_{\tau p} M_2 \quad \Leftrightarrow \quad M_1 \sim_\tau M_2.$$

Now we are ready to define a resource τp-similarity:

Definition 10. *Let $N = (P, T, F, l)$ be a saturated labelled Petri net with invisible transitions. Resources r and s are called τp-*similar *(denoted $r \approx_{\tau p} s$) iff for every marking R, $r \subseteq R$ implies $R \sim_{\tau p} R - r + s$.*

From the equality of $\sim_{\tau p}$ and $\sim_\tau$ in saturated nets we immediately have:

Corollary 1. *Let $N = (P, T, F, l)$ be a saturated labelled Petri net with invisible transitions, $r, s \in \mathcal{M}(P)$. Then*

$$r \approx_{\tau p} s \quad \Leftrightarrow \quad r \approx_\tau s.$$

So, in saturated nets it is sufficient to look for τp-similarities.

Definition 11. *Let $N = (P, T, F, l)$ be a saturated labelled Petri net with invisible transitions. An equivalence relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ is called a* resource τp-bisimulation *if B^{AT} is a marking τp-bisimulation.*

In the case of τp-relations all basic properties also hold:

Proposition 5.
1. *Resource τp-similarity is closed under addition and transitivity; hence it has finite AT-basis.*
2. *Resource τp-similarity is undecidable.*
3. *If $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ is a resource τp-bisimulation and $(r_1, r_2) \in B$ then $r_1 \approx_{\tau p} r_2$.*
4. *If $B_1, B_2 \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ are resource τp-bisimulations then $B_1 \cup B_2$ is a resource τp-bisimulation;*
5. *For any N there exists the largest resource τp-bisimulation (denoted by $B_{\tau p}(N)$), and it is an equivalence.*

Proof.

1) Immediately from the definition of resource τp-similarity.

2) From Cor. 1 and the undecidability of $(\approx_\tau)$.

3) Immediately from the definitions.

4) The proof is almost the same as in Prop. 3: the only difference is that we consider not an imitating transition but an imitating parallel step.

5) Note that we can take a union of all resource τp-bisimulations.

Definition 12. *Let $N = (P, T, F, l)$ be a saturated labelled Petri net with invisible transitions. We say that a relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ conforms to the* weak τp-transfer property *if for all $(r, s) \in B$, $t \in T$ s.t. ${}^\bullet t \cap r \neq \emptyset$, there exists an imitating parallel step $U \in \mathcal{M}(T)$ s.t. $l(t) =_\tau l(U)$ and, denoting $M_1 = {}^\bullet t \cup r$ and $M_2 = {}^\bullet t - r + s$, we have $M_1 \xrightarrow{t} {M_1}'$ and $M_2 \xrightarrow{U} {M_2}'$ with $(M_1', M_2') \in B^{AT}$.*

In saturated nets the weak τp-transfer property is a necessary and sufficient condition for its extended version, which guarantees the imitation of a parallel step rather than a single transition:

Definition 13. *Let $N = (P, T, F, l)$ be a saturated labelled Petri net with invisible transitions. We say that a relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ conforms to the* extended weak τp-transfer property *if for all $(r, s) \in B$ and any parallel step $V \in \mathcal{M}(T)$ s.t. ${}^\bullet V \cap r \neq \emptyset$, there exists an imitating parallel step $U \in \mathcal{M}(T)$ s.t. $l(V) =_\tau l(U)$ and, denoting $M_1 = {}^\bullet V \cup r$ and $M_2 = {}^\bullet V - r + s$, we have $M_1 \xrightarrow{V} {M_1}'$ and $M_2 \xrightarrow{U} {M_2}'$ with $(M_1', M_2') \in B^{AT}$.*

Lemma 2. *Let $N = (P, T, F, l)$ be a saturated labelled Petri net with invisible transitions. The relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ conforms to the weak τp-transfer property iff it conforms to the extended weak τp-transfer property.*

Proof. ($\Leftarrow$) Since the weak transfer property is a special case of the extended weak transfer property.

($\Rightarrow$) Assume the converse: the extended property does not hold, so there exists $(M_1, M_2) \in B^{AT}$, $V = \{t_1, \ldots, t_k\} \in \mathcal{M}(T)$ with $M_1 \xrightarrow{V} M_1'$, s.t. there exists no imitating parallel step $U \in \mathcal{M}(T)$ with the same visible label $l(V) =_\tau l(U)$ and $M_2 \xrightarrow{U} M_2'$ and $(M_1', M_2') \in B^{AT}$.

Consider the transition firing $M_1 \xrightarrow{t_1} M_1^1$. From the weak τp-transfer property it follows that this transition has an imitating parallel step $M_2 \xrightarrow{W_1} M_2^1$ such that $(M_1^1, M_2^1) \in B^{AT}$.

Note that $V = \{t_1, \ldots, t_k\}$ is a parallel step at marking M_1, hence after the firing of one of these transitions all other are still enabled. Therefore we can repeat the previous reasoning for the new pair of markings $(M_1^1, M_2^1) \in B^{AT}$ and transition t_2. And continue this until t_k:

$$\begin{array}{ccc}
M_1 & B^{AT} & M_2 \\
t_1 \downarrow & & \downarrow W_1 \\
M_1^1 & B^{AT} & M_2^1 \\
t_2 \downarrow & & \downarrow W_2 \\
M_1^2 & B^{AT} & M_2^2 \\
t_3 \downarrow & & \downarrow W_3 \\
\dots & & \dots \\
t_k \downarrow & & \downarrow W_k \\
M_1' = M_1^k & B^{AT} & M_2^k = M_2'
\end{array}$$

At the end we got a sequence of parallel steps

$$M_2 \stackrel{W_1}{\to} M_2^1 \stackrel{W_2}{\to} M_2^2 \stackrel{W_3}{\to} \dots \stackrel{W_k}{\to} M_2^k = M_2',$$

imitating the firing of parallel step $M_1 \stackrel{V}{\to} M_1'$. The net is saturated so for any sequence of transitions (note that a parallel step also can be considered as a sequence of transitions) there exists an imitating parallel step with the same label, precondition and postcondition ($M_2 \stackrel{U}{\to} M_2'$) – q.e.d.

Note that, unlike the weak transfer property, the extended weak transfer property can not be effectively checked by the search of resource pairs, since the set of parallel steps is infinite.

Theorem 5. *Let $N = (P, T, F, l)$ be a saturated labelled Petri net with invisible transitions. An equivalence relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ conforms to the weak τp-transfer property iff B is a resource τp-bisimulation.*

Proof. ($\Leftarrow$) Since the weak τp-transfer property is the τp-transfer property for a bounded (finite) subset of pairs of resources.

($\Rightarrow$) The proof is similar to the proof of Th. 4, with the additional use of Lm. 2.

Assume the converse: let B^{AT} does not conform to the τp-transfer property, i.e. there exist $(M_1, M_2) \in B^{AT}$, $t \in T$ with $M_1 \stackrel{t}{\to} M_1'$, s.t. there are no imitating parallel step $U \in \mathcal{M}(T)$ with $l(t) = l(U)$, $M_2 \stackrel{U}{\to} M_2'$ and $(M_1', M_2') \in B^{AT}$.

Consider a pair of markings $(M_1, M_2) \in B^{AT}$. From Lm. 1 this pair can be obtained by a transitive closure of several pairs from B^A (additive closure of B):

$$(H_1, H_2), (H_2, H_3), \dots, (H_{k-1}, H_k) \in B^A, \text{ where } H_1 = M_1, H_k = M_2.$$

Consider the pair (H_1, H_2).

$$(H_1, H_2) = (r_1 + r_2 + \dots + r_l, s_1 + s_2 + \dots + s_l), \text{ where } (r_i, s_i) \in B$$

$H_1 = {}^\bullet t \cup r_1 + F_1$. Due to the weak transfer property for the pair (r_1, s_1) there exists an imitating parallel step $V \in \mathcal{M}(T)$ s.t. $l(t) = l(V)$,${}^\bullet t \cup r_1 \xrightarrow{t} G_1$ and ${}^\bullet t - r_1 + s_1 \xrightarrow{V} G_2$, where $(G_1, G_2) \in B^{AT}$.

Since ${}^\bullet t \cup r_1 \subseteq H_1$, we can add the resource $F = H_1 - {}^\bullet t \cup r_1$ to preconditions and postconditions:

$$
\begin{aligned}
{}^\bullet t \cup r_1 + F &\xrightarrow{t} G_1 + F \\
{}^\bullet t - r_1 + s_1 + F &\xrightarrow{V} G_2 + F
\end{aligned}
$$

From the reflexivity of B and the additive closure of B^{AT} the new pair of markings is also decomposable by $B : (G_1 + F, G_2 + F) \in B^{AT}$.

We obtained a new marking $H_1' = {}^\bullet t - r_1 + s_1 + F = H_1 - r_1 + s_1$. Note that it still contains $r_2 + \ldots + r_l$. Therefore, we can apply the same reasoning one more time, replacing resource r_2 by the bisimilar resource s_2, now using Lm. 2 and constructing an imitating parallel step not for a transition but for a parallel step V.

Apply this $l - 1$ times. Using transitive closure of B^{AT}, at the end we obtain a parallel step W that can imitate t at marking H_2.

Now proceed to the next pair (H_2, H_3) and repeat the procedure for the parallel step W. And so on, until the last pair (H_{k-1}, H_k). Finally we obtain a parallel step U that can imitate t at marking $H_k = M_2$.

Thus, in saturated nets the weak τp-transfer property can be used in the construction of resource τp-bisimulation.

3.5 Approximation

As in ordinary Petri nets, in the case of saturated (saturable) nets with invisible transitions there is a way of constructing an approximation of the maximal resource τp-bisimulation. If we consider not an infinite set of network resources, but only it's finite subset, then it will be possible to check the weak τp-transfer property.

Let $N = (P, T, F, l)$ be a saturated labelled Petri net with invisible transitions, $q \in Nat$ — some parameter. By $\mathcal{M}_q(P)$ we denote the set of all resources, containing not more then q tokens in the net: $\mathcal{M}_q(P) = \{r \in \mathcal{M}(P) \ : \ |r| < q\}$.

The largest resource τp-bisimulation on $\mathcal{M}_q(P)$ is defined as the union of all resource τp-bisimulations on $\mathcal{M}_q(P)$. We denote it by $B_{\tau p}(N, q)$. Since $\mathcal{M}_q(P)$ is finite, we can use the weak transfer property to compute $B_{\tau p}(N, q)$.

Definition 14. *(Underapproximation of largest resource τp-bisimulation)*

Input: *A saturated labelled Petri net with invisible transitions $N = (P, T, F, l)$, parameter $q \in Nat$.*

Output: *Relation $B_{\tau p}(N, q)$.*

Step 1: *Let $C = \{(\emptyset, \emptyset)\}$ — an empty set of pairs (considered as a relation over $\mathcal{M}_q(P)$).*

Step 2: *Compute* $B = (\mathcal{M}_q(P) \times \mathcal{M}_q(P)) \setminus C$. *Since* $\mathcal{M}_q(P)$ *is finite the set of pairs* B *is also finite.*

Step 3: *Compute* B_s *— the ground basis of* B.

Step 4: *Check, whether* B_s *conforms to the weak* τp*-transfer property: it is sufficient to test all non-reflexive elements of* B_s.

- *If all pairs conforms to the weak* τp*-transfer property then stop and return* B *— the bisimulation.*
- *Otherwise there are* $(r, s) \in B_s^{nr}$ *and* $t \in T$ *with* ${}^\bullet t \cap r \neq \emptyset$*, s.t. the firing* $M_1 \xrightarrow{t} M_1{}'$ *with* $M_1 = {}^\bullet t \cup r$ *can not be imitated by a parallel step with the same label and with precondition* $M_2 = {}^\bullet t - r + s$. *In this case add* (r, s) *and* (s, r) *to* C *and go back to Step 2.*

For any marking the set of active parallel steps is finite. Also note that the set $\mathcal{M}_q(P) \times \mathcal{M}_q(P)$ is finite. Hence the algorithm always stops.

Denote by $\mathcal{R} = |\mathcal{M}_q(P)|$ the size of the set of considered resources.

At the Step 2 we search through the set of all parallel steps with at most one visible label, that can fire at marking M_2. Each invisible transition can participate in the parallel step at most $|M_2|$ times, since it uses at least one input token.[1] There is also at most one visible transition. Hence we have to check at most $|T||M_2|^{|T|}$ multisets of transitions.

The size of marking $M_2 = {}^\bullet t - r + s$ can be evaluated as $\mathrm{O}(|s|) = \mathrm{O}(q)$.

Using our previous estimations of complexity for ground basis calculation (polynomial w.r.t. $\mathcal{R}$) and the complexity of other steps of algorithm (polynomial w.r.t. the size of the net), we obtain the overall complexity of

$$\mathrm{O}(\max\{|P|\ \mathcal{R}^9, |T|^2 q^{|T|} |P|\ \mathcal{R}^7\}).$$

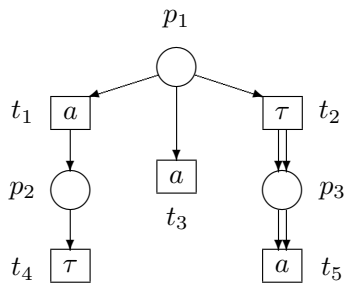
So in the case of nets with invisible transitions the complexity of the algorithm increased significantly (the linear dependence on $|T|$ was replaced by an exponential one). Such a jump is explained by the transition from sets of transitions to multisets.

Consider an example of calculations (Fig. 6). With $q = 1$ we found that resource p_2 is τp-similar to an empty resource (i.e. the place p_2 is redundant). Increasing the parameter ($q = 2$), we obtained one more pair of similar resources $p_1 \approx_{\tau p} 2p_3$.

4 Conclusion

The proposed method for finding pairs of similar resources is of particular interest for certain applications. In addition, the use of resource bisimulation allows one to reduce a Petri net with conservation of its behavior. This reduction is

[1] Without loss of generality we can assume that a net contains no invisible transitions with empty preconditions, since such transitions are redundant and can always be removed from the net along with places, included in their postconditions.



$q = 1: \quad p_2 \approx_{\tau p} \emptyset$

$q = 2: \quad p_1 \approx_{\tau p} 2p_3$

Fig. 6. An example of approximation: resource τp-bisimulation of a saturated Petri net with invisible transitions

important when analyzing properties of the Petri net, since the computational complexity of the majority of algorithms used in analysis depends exponentially on the size of the net.

Important open questions concern decidability and complexity of related algorithmic problems. For example, we have already shown that all types of resource similarity (ordinary, τ-, τp-) are undecidable. On the other hand, the problem of $B(N)$ (and $B_\tau(N)$, and $B_{\tau p}(N)$) computability is still open. We have introduced only the underapproximations.

Acknowledgment. I would like to express my sincere gratitude to my mentor Irina Lomazova for her support and encouragement.

References

1. Autant, C., Pfister, W., Schnoebelen, Ph.: Place bisimulations in Petri nets. Proc. of ATPN'92. Lecture Notes in Computer Science, 616, 45–61 (1992)
2. Autant, C., Pfister, W., and Schnoebelen, Ph.: Place bisimulations for the reduction of labeled Petri nets with silent moves. Proc. of ICCI'94, Peterborough, Ontario (1994)
3. Bashkin, V. A., Lomazova, I. A.: Reduction of Coloured Petri nets based on resource bisimulation. Joint Bulletin of NCC & IIS (Comp. Science), 13, 12–17 (2000)
4. Bashkin, V. A., Lomazova, I. A.: Petri Nets and resource bisimulation. Fundamenta Informaticae, 55(2), 101–114 (2003)
5. Bashkin, V. A., Lomazova, I. A.: Resource similarities in Petri net models of distributed systems. Proc. of PACT'2003. Lecture Notes in Computer Science, 2763, 35–48 (2003)
6. Hirshfeld, Y.: Congruences in commutative semigroups. Research report ECS-LFCS-94-291, Department of Computer Science, University of Edinburgh (1994)
7. Jančar, P.: Decidability questions for bisimilarity of Petri nets and some related problems. Proc. of STACS'94. Lecture Notes in Computer Science, 775, 581–592 (1994)
8. Lomazova, I. A.: Resource Equivalences in Petri Nets. Proc. of PETRI NETS 2017. Lecture Notes in Computer Science, 10258, 19–34 (2017)
9. Milner, R. A Calculus of Communicating Systems. Springer Berlin Heidelberg (1980)

10. Park, D. Concurrency and automata on infinite sequences. Theoretical Computer Science. Lecture Notes in Computer Science, 104, 167–183 (1981)
11. Redei, L.: The theory of finitely generated commutative semigroups. Oxford University Press, New York (1965)
12. Shnoebelen, Ph., Sidorova, N.: Bisimulation and the reduction of Petri nets. Proc. of ATPN'2000. Lecture Notes in Computer Science, 1825, 409–423 (2000)

Compatibility Control of Asynchronous Communicating Systems with Unbounded Buffers

D. Dahmani[1], M.C. Boukala[1], H. Mountassir[2], and S. Chouali[2]

[1] MOVEP, Comp. Sci. Dept, USTHB, Algiers.
dzaouche,mboukala@usthb.dz,
[2] Femto-ST/DISC, Comp. Sci. Dept, Burgundy, Franche-Comté University
hmountassir,schouali@femto-st.fr

Abstract. The composition of heterogeneous software components is required in many domains to build complex systems. However, such compositions raise mismatches between components such as unspecified messages. Checking compatibility for asynchronously communicating systems with unbounded channels is undecidable. In this paper, we propose a compatibility control approach based on a coverability product, which is a finite abstraction of the asynchronous products of I/O-transition systems. This coverability product is used to check UR-compatibility, without requiring a synchronizabity of the peers. We distinguish between messages brought by acyclic path and those brought by cycles. This allows us to overcome over-approximation for some arcs. Furthermore, we define relationships, called patterns, to check a good choreography between peers in terms of emission and reception activities.

Keywords: communicating systems, compatibility analysis, coverability product, patterns, infinite systems.

1 Introduction

Component-based development aims at facilitating the construction of very complex applications by reusing and composing existing components. The assembly of components offers a great potential for reducing cost and time to build complex software systems and improving system maintainability and flexibility. A software component is generally developed independently and is assembled with other components, which have been designed separately, to create complex systems. Normally glue code is written to realize such an assembly. Compatibility checking is used to control if composed components can work together *without errors*. This verification is very important for ensuring correct interaction between components and may be done by using different formal models such as interface automata, I/O-transition systems, π-calcul, Petri nets and state machines [1, 8, 9, 12].

There are several incompatibility notions: (1) *label incompatibility* concerns messages exchanged between components which can be handled differently. For

instance, messages are named differently, methods are called with parameters which don't match perfectly (incompatible types, different orders, ...) [1]. (2) The *bidirectional notion* requires that when a component sends a message, there is another component which eventually receives it, and when a component is waiting to receive a message, there is another which must send that message. (3) The *unspecified reception* (UR-compatibility) is less restrictive than the *bidirectional notion*, since the reception of messages expected is not required for *unspecified reception* notion. The UR-compatibility requires that the composition of components doesn't contain any deadlock, i.e. starting from their initial states, all components can either evolve or terminate if they are in final states and their buffers are empty [1, 6].

To deal with incompatible components, pessimistic and optimistic approaches have been proposed. The pessimistic approach considers that two components are compatible if they can always work together in any environment while, in the optimistic approach, two components are compatible if they can be used together in some helpful environment [1].

Components can interact synchronously or asynchronously. In synchronous communications, a send action is a lock-step, since it is allowed only when the receiver is ready to perform the corresponding reception. Thus, send and receive actions are performed simultaneously. In asynchronous communications, a send action is not a lock-step. The messages sent are added to the receiver buffers. Such buffers may be ordered or unordered. Analyzing asynchronous communicating components with unbounded buffers is a complex task, undecidable in general [10, 13], since it may lead to an infinite state space. Several works aiming at analyzing such systems bound the size of buffers or the number of iterations per cycles. Bounding such parameters is not a good solution since new unexpected errors can occur when the values of parameters change or exceed the fixed bounds. Thus, the main problem of these approaches is to choose the appropriate bounds to study the compatibility of the infinite communicating systems. In this paper, we propose a compatibilty control approach, based on a coverability product, for asynchronous communicating components with *unordered* and *unbounded* buffers.

2 Related works

Brand and Zafiropulo are among the first to have proposed works in the area of service compatibility checking [6]. They use communicating finite state machines to model interacting processes executed in parallel and exchanging messages via FIFO buffers. Their works focus on unspecified receptions compatibility notion for interaction protocols. When considering unbounded buffers, the authors show that the resulting state spaces may be infinite, and the problem becomes undecidable.

The works in [7, 8] use Petri net models to treat incompatibility problems. Some works rely on extensions of Petri nets, like open nets to model communicating processes, assuming asynchronous communication over non-ordered message buffers. As far as asynchronous semantics is considered, compatibility analysis has been proven to be undecidable for unbounded open nets. The authors deal only with limited-communications which give bounded open nets. In [12] authors use the so-called state equation, which is based on the standard linear programming in Petri nets, to find a necessary but not sufficient condition for compatibility control.

Recently, Bouajjani and Emmi [5] consider a bounded analysis of asynchronous message-passing programs with ordered message queues. Their approach does not limit the number of communicating processes nor the buffers' size. However, the *number of iterations of communication cycles* is limited. Despite the potential for huge exploration of unbounded process contexts, the proposed bounding scheme gives rise to a simple and efficient program analysis by reduction to sequential programs.

In [13], an approach on checking the compatibility of peers communicating asynchronously by message exchange over unbounded buffers is proposed. The approach requires that peers are synchronisable. In this case, the synchronous system behaves like the asynchronous one for any buffer size. Thus, the compatibility check on the asynchronous version of the system is reduced to the synchronous version, which is finite and decidable. However, the approach cannot conclude anything about non synchronizable peers. In [11], authors focus on the verification of weak asynchronous compatibility relying on half-duplex systems instead of synchronizability and provide a decidable criterion that ensures weak asynchronous compatibility.

3 Motivation

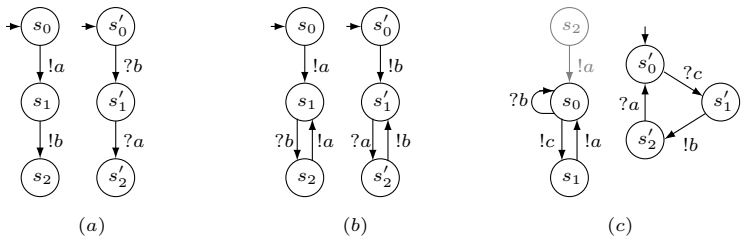


Fig. 1: Asynchronous communicating peers

Figure 1.a shows a simple example which highlights the relevance of unordered queues in a context of asynchronous communication. Peer 1 sends message a followed by b, but peer 2 consumes the messages in a reverse order of

their emission. The peers are compatible with unordered queues but not with FIFO queues. In Figure 1.b, peer 1 (resp. peer 2) sends message a (resp. b) and loops infinitely by consuming b (resp. a) and then sending a (resp. b). All works based on synchronizability property [2, 3, 13] cannot conclude anything about the compatibility of such a system, since peers are not synchronizable. However, they are free of deadlock and any message sent is consumed. Consider Figure 1.c and suppose that s_0 is the initial state of the left peer. This latter can either loop on the emission of c followed by a, or the reception of b. Peer 2 can indefinitely receive c, send b and then receive a. Despite the perfect coherence between the production and consumption patterns of these peers, there is no conclusion about their compatibility based on synchronizability property.
In this paper, we propose an approach to control the compatibility of asynchronous communicating peers without requiring the synchronizability property. Our solution is mainly based on the construction of a *coverability asynchronous product* which is always finite. Some analysis techniques are also proposed to overcome the over-approximation of a coverability asynchronous product and check the coherence of peers's production and consumption patterns in their cyclic stages.

4 Coverability product

4.1 I/O-transition systems

First we give the definition of an I/O-transition system.

Definition 1 *An I/O-transition system A is a 4-tuple* $(S_A, S_A^{init}, \Sigma_A, \tau_A)$ *such that:*

1. S_A *is a finite set of states.*
2. $S_A^{init} \subset S_A$ *a set of initial states.*
3. $\Sigma_A = \Sigma_A^I \cup \Sigma_A^O \cup \Sigma^H$*, where* Σ_A^I *,* Σ_A^O *and* Σ_A^H *are finite disjoint sets of input, output and internal actions.*
4. $\tau_A \subseteq S_A \times \Sigma_A \times S_A$ *is a set of steps.*

In an I/O-transition system, each input (resp. output, internal) action is preceded by the symbol ? (resp. !, ;). A finite execution σ in an I/O-transition system is an alternating sequence of states and actions $s_0 \xrightarrow{a_1} s_1 \ \ldots \ \xrightarrow{a_n} s_n$ such that $s_i \xrightarrow{a_{i+1}} s_{i+1} \in \tau_A$. For the sake of clarity, we suppose that S_A^{init} is reduced to a single state.

4.2 Coverability product construction

The asynchronous product of I/O-transition systems with unbounded buffers may be infinite. In this paper, we propose to build a *finite* coverability product, inspired from the standard approach of constructing a coverability graph for P/T

nets [4]. The coverability graph covers all reachable states where ω, a specific symbol, is used to represent reachable states with unbounded messages.

In our approach, we distinguish between the message occurrences brought by elementary paths and those brought by cycles. The former are explicitly represented in the product, while the latter are abstracted by a set Ω. Thus, Ω gives the set of unbounded messages. In the sequel, we confuse between terms message and action.

Let $A_1 = (S_{A_1}, s_{0A_1}, \Sigma_{A_1}, \tau_1)$ and $A_2 = (S_{A_2}, s_{0A_2}, \Sigma_{A_2}, \tau_{A_2})$ be two I/O-transition systems. Automata A_1 and A_2 fulfils the following conditions: $\Sigma^I_{A_1} \cap \Sigma^O_{A_2} = \emptyset$ and $\Sigma^O_{A_1} \cap \Sigma^I_{A_2} = \emptyset$. Let $\Sigma = \Sigma_{A_1} \cup \Sigma_{A_2}$ be the set of actions of A_1 and A_2.

Definition 2 *The coverability product of A_1 and A_2, denoted by $A_1 \otimes A_2$, is a 7-tuple $(V, \Pi_1, \Pi_2, \tau, M, \Omega, v_0)$. V is the set of nodes, with $v_0 \in V$ being the root. Π_i (with $i = 1,2$) is a function from V to S_{A_i}. $\tau \subseteq V \times \Sigma \times V$ is the set of labelled arrows. M is a function from $V \times \Sigma$ to $\mathbb{N}$, assigning an ordinary marking to every node. Ω is a function from V to 2^Σ giving, for a node v, the set of its infinite actions (messages).*

A step of $A_1 \otimes A_2$ is either a step of A_1 or A_2. The following definitions are useful to build $A_1 \otimes A_2$.

Definition 3 *(Enabling condition) A state s of S_{A_i}, for $i \in \{1, 2\}$, enables a node v of $A_1 \otimes A_2$ iff $\exists s \xrightarrow{\delta a} s' \in \tau_{A_i}$ and $\Pi_i(v)$=s and either:*

- $\delta \in \{!\ ,\ ;\}$,
- *or* $\delta = ? \Rightarrow (a \in \Omega(v) \vee M(v)(a) > 0\ )$.

Definition 4 *Let u and v be two nodes of $A_1 \otimes A_2$. Two nodes u and v are equal, denoted by $u = v$, iff $\Pi_1(u) = \Pi_1(v)$, $\Pi_2(u) = \Pi_2(v)$, $\forall a \in \Sigma$, $M(u)(a) = M(v)(a)$ and $\Omega(u) = \Omega(v)$.*

Definition 5 *Let u and v be two nodes of $A_1 \otimes A_2$. Node u is less than v , denoted by $u < v$, iff $\Pi_1(u) = \Pi_2(v)$, $\Pi_2(u) = \Pi_2(v)$, $\exists a \in \Sigma$, $M(u)(a) < M(v)(a)$, $\forall b \in \Sigma$, $(M(u)(b) \leq M(v)(b)$ or $b \in \Omega(u))$ and $\Omega(u) \subseteq \Omega(v)$.*

Given two I/O-transition systems A_1 and A_2, the algorithm 1 constructs the coverability product $A_1 \otimes A_2$. For the sake of clarity we present the product for two I/O-transition systems. However, the algorithm can be easily extended to n I/O-transition systems.

Algorithm 1: Coverability graph construction

Data: Two I/O-transition systems $A_1 = (S_{A_1}, s_{0A_1}, \Sigma_{A_1}, \tau_1)$ and $A_2 = (S_{A_2}, s_{0A_2}, \Sigma_{A_2}, \tau_{A_2})$

Result: Coverability Graph $G = (V, \Pi_1, \Pi_2, \tau, M, \Omega, v_0)$

```
v_0 ← new node ;
V ← {v_0} ;
Π_1(v_0) = s_{0A_1}; Π_2(v_0) = s_{0A_2} ;
∀a ∈ Σ, M(v_0)(a)=0 and Ω(v_0) = ∅ ;
τ ← ∅ ;
unprocessed ← {v_0};
while unprocessed ≠ ∅ do
    v ← element of unprocessed;
    foreach s -δa-> s' of τ_{A_1}, s.t. state s enables node v do
        v' ← new node ; Π_1(v') = s'; Π_2(v') = Π_2(v) ; Ω(v') = Ω(v) ;
        if δ =! then M(v')(a) = M(v)(a) + 1 ;
        if δ =? ∧ M(v)(a) > 0 then M(v')(a)=M(v)(a) − 1 ;
        if δ =? ∧ M(v)(a) = 0 ∧ a ∈ Ω(v) then M(v')(a)=M(v)(a) ;
        if δ =; then M(v')(a)=M(v)(a) ;
        /* x ⇝_τ y means a path from x to y with arcs in τ.    */
        if ∃u ⇝_τ v, with u < v' then
            foreach a ∈ Σ do
                if M(u)(a) < M(v')(a) then
                    M(v')(a) = M(u)(a) ;
                    Ω(v') = Ω(v') ∪ {a} ;
                end
            end
        end
        if ∄w ∈ V s.t. w = v' then
            V ← V ∪ {v'} ;
            unprocessed ← unprocessed ∪ {v'} ;
            τ ← τ ∪ {v -δa-> v'} ;
        else
            select w ∈ V s.t. w = v' ;
            τ ← τ ∪ {v -δa-> w}
        end
    end
    /* This loop is repeated for each edge s -δa-> s' of τ_{A_2}.    */
    unprocessed ← unprocessed \ {v} ;
end
```

4.3 Reduction of I/O-transition systems

The interleaving between some actions of A_1 and A_2 are not necessary for the properties we target (dealock and UR-reception). They only increase the size of $A_1 \otimes A_2$. For instance, from state s_0 of Figure 1.c, message c is sent and then message a. These actions can be done by the left peer without any interaction with the right peer. Hence, they can be replaced by a single and an atomic action $!ca$. Furthermore, the path $s_0 \xrightarrow{!c} s_1 \xrightarrow{!a} s_0$ is reduced to $s_0 \xrightarrow{!ca} s_0$. This transition produces one occurrence of message a and another of b. We can also reduce sequences of internal or/and emission actions. However, the reduction of a sequence $s_j? \xrightarrow{\delta_0 a_0} \ldots \xrightarrow{\delta_k a_{k-1}} s_{j+k}$ of A_i, $i \in \{1,2\}$, is not always possible. Conditions required to reduce a path are given below:

1. **No interaction**: $\delta_0, \ldots \delta_{k-1} \in \{!,;\}$.
2. **No intermediary branching states**: $\forall j < h < j+k$, s_h is not a branching state.
3. **Occurrence in an elementary cycle**: Either all the arcs of the sequence are in an elementary cycle or all are not.

The first point states that there is no interaction between the peer and its correspondent, whereas the two other points allow to preserve the behavior of the peer.

Example 1 *Consider again the I/O-transition systems depicted in Figure 1.c and suppose that s_2 is the initial state of the left peer. Only the left automaton can be reduced by replacing the edges $s_0 \xrightarrow{!c} s_1$ and $s_1 \xrightarrow{!a} s_0$ by $s_0 \xrightarrow{!ca} s_0$. The coverability product, constructed by Algorithm 1, is shown in Figure 2. Unbounded actions of a node v, $\Omega(v)$ are given between brackets. The occurrences of actions brought by elementary path are explicitly represented. In the reachable state $(s_0,s_0',a,\{c,a\})$, say u, the current state $\Pi_1(u)$ of the first (resp. $\Pi_2(u)$ of the second) I/O-transition system is s_0 (resp. s_0'). The marking M of u is reduced to one occurrence of a ($M(u)(a)$=1, $M(u)(b)$=0 and $M(u)(c)$=0) and $\Omega(u) = \{c,a\}$ meaning that the occurrences a and c are potentially infinite.*

Non-travial scc are bordred by dashed boxes and $C1$ is terminal scc. Some arcs are represented by gray lines to highlight the over-approximation notion,- which will be introduced in the following section.

Notation 1 *A strongly connected component (scc) C of a graph G is a maximal set of nodes $C \subseteq S_G$, where S_G is the set of nodes of graph G, such that for every pair of nodes u and v in C, there is a directed path from u to v and a directed path from v to u. A scc C composed of one node is said trivial, otherwise non trivial. A path or cycle in G is called elementary if no node occurs more than once. A cycle is called simple if no edge occurs more than once. A scc is terminal if it has no outgoing edge. Let σ be a sequence of transitions or an elementary cycle of $A_1 \otimes A_2$. $\#(\sigma,!a)$ (resp. $\#(\sigma,?a)$) gives the number of occurrences $!a$ (resp. $?a$) in σ and $\#(\sigma,a)$ represents $\#(\sigma,!a) - \#(\sigma,?a)$.*

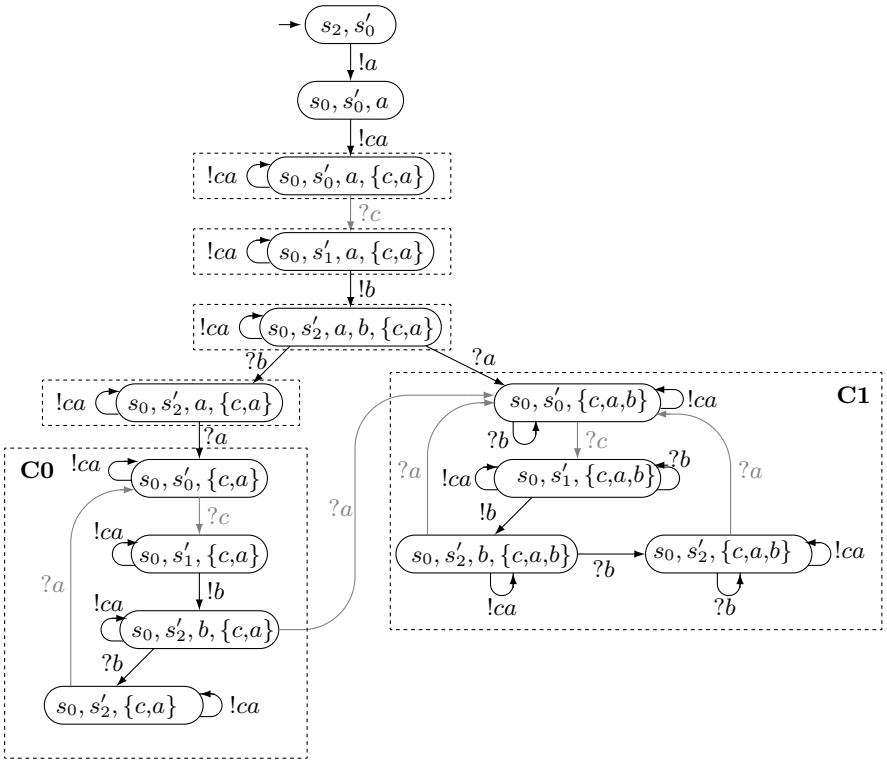


Fig. 2: Coverability product for peers of Figure 1.c

Definition 6 *The set of actions which may* ***accumulate*** *in the nodes of a cycle μ, denoted by $\Psi(\mu,!)$, corresponds to $\{a \in \Sigma \mid \#(\mu,a) > 0\}$. A cycle μ may consume accumulated actions, those in $\Psi(\mu,?) = \{a \in \Sigma \mid \#(\mu,a) < 0\}$.*

In Figure 2, each cycle μ labelled by $!ca$ is an infinite producer of c and a ($\#(\mu,c) = 1$ and $\#(\mu,a) = 1$). Similarly, for instance, cycle $\mu = ?c\ !b\ ?b\ ?a$ consumes actions c and a which are not produced by μ ($\#(\mu,c) = -1$ and $\#(\mu,a) = -1$) but accumulated by cycles labelled by $!ca$.

4.4 Over-approximation of coverability product

We give in Figure 3.b a part of the coverability product of peers in 3.a. In node v_1, peer A_2 does not block, since peer A_1, at state s_0, is able to provide messages a. Now consider the sequence $v_0 \xrightarrow{!a} v_1 \xrightarrow{!a} v_1 \xrightarrow{?a} v_2 \xrightarrow{!b} v_3 \xrightarrow{?b} v_4$. At the end of this sequence, peers A_1 and A_2 are in states s_1 and s'_2, respectively. Observe that A_2 waits indefinitely message a which will never be provided by peer A_1. Indeed, at state s_1, peer A_1 cannot produce message a. Hence, node v_4, contray to v_1, is a potential deadlock for peer A_2.

Both edges $v_1 \xrightarrow{?a} v_2$ and $v_4 \xrightarrow{?a} v_5$ are over-approximated. However, the first is always possible, whereas the second is only possible for some executions. An over-approximated edge $v \xrightarrow{?a} v'$ deserves special attention, since, for some executions, node v' is *never reached*. Hence, their presence makes $A_1 \otimes A_2$ not reliable. A loop $v \xrightarrow{?a} v$ is not considered as over-approximated even if $M(v)(a) = 0$.

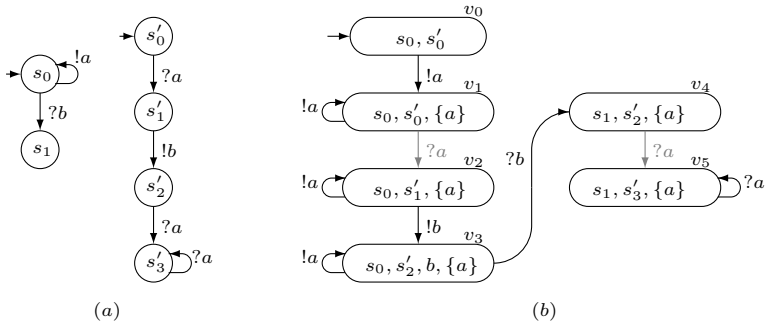


Fig. 3: Over-approximation

In this section, we propose a technique aiming at discarding gradually the over-approximated status for some edges. Algorithm 2 summarises this technique. Its input is a strongly connected component C of $A_1 \otimes A_2$. The algorithm returns *false* whenever some edges remain over-approximated, otherwise it returns *true*. The algorithm uses the sets *simpleCycles(C)* and *reliableCycles(C,v)*, with v a node of C. The former contains all simple cycles of C and the second is a subset of $simpleCycles(C)$ where each cycle contains node v and is composed of non over-approximated edges only.

The algorithm builds a set of nodes called V_{over}. A node of V_{over} is an initial extremity of at least one over-approximated edge. The algorithm is iterative. At each step, V_{over} is computed. The over-approximated status of each edge $v \xrightarrow{?a} v'$, with $v \in V_{over}$ is discarded whenever there is a reliable cycle $\mu \in$*reliableCycles(C,v)* producing indefinitely action a, i.e. $a \in \Psi(\mu,!)$. At the next step, V_{over} is updated, since some edges turn into non over-approximated. Hence, new cycles become reliable.

The termination of the algorithm is ensured since there is a finite number of simple cycles, actions and edges. The algorithm is applied to the strongly connected components of $A_1 \otimes A_2$. It is obvious that if the product contains no over-approximated edges, then it is reliable and can be used to check the free of deadlock and UR-compatibility properties.
Consider again the coverability product of Figure 2. Applying the precedent algorithm to non trivial scc ($C1$) induces the following steps.

1. Gray edges (over-approximated) of $C1$ compose E_{over}.

Algorithm 2: dealOverapproxation

```
Data: a strongly connected components C of A1 ⊗ A2
Result: boolean
new =true;
E_over = {v --?a--> v' an edge of C s.t. M(v)(a) = 0 and v ≠ v'} ;
/* E_over contains the set of over-approximated edges                    */
∀μ ∈ simpleCycles(C), μ.visited = false ;
while new do
    V_over = {v a node C | ∃v --?a--> v' ∈ E_over};
    new=false ;
    foreach v ∈ V_over do
        actions = {a ∈ Σ | a ∈ Ψ(μ,!) ∧ μ ∈ reliableCycles(C,v) ∧ ¬μ.visited} ;
        foreach μ ∈ reliableCycles(C,v) do
            μ.visited = true
        end
        foreach over-approximated edge e = v --?a--> v' do
            if a ∈ actions then
                E_over = E_over \ {e} ;
                new = true;
            end
        end
    end
end
if E_over ≠ ∅ then return false else return true;
```

2. In the first iteration, V_{over} contains the three nodes of $C1$ which are initial extremities of gray arcs. A reliable simple cycle, labelled by $!ca$, loops around each node of V_{over}. Thus, the set of actions $\{c,a\}$ is associated to each node of V_{over}. All gray edges are then removed from E_{over}.
3. In the second iteration, V_{over} is empty and there is no change about edge status, inducing the termination of the algorithm. The algorithm returns true.

Likewise, the algorithm 2 is applied to the other scc of $A_1 \otimes A_2$ depicted in Figure 2. We can easily see that the coverability product depicted in Figure 2 is free of deadlock.

5 UR compatibility verification

In this section, we focus on the widely notion unspecified receptions UR. A set of peers is UR-compatible if they do not deadlock and each sent message by a peer is received by another one. Consider the peers given in Figure 1.c and consider s_0 as initial state of the left peer. It is obvious that there are a good *choreography* and *proper* interactions between the peers, since the right peer

requires an equality between messages a and c, a relation satisfied by the left peer of the Figure 1.c. However, when we add a cycle, for instance, $s_0 \xrightarrow{!a} s_0$ for the left peer, the equality relation between a and c is lost inducing a mess in the consumption activity. In this paper, we propose an approach to determine relationships between unbounded actions and use such relationships to check the UR compatibility through $A_1 \otimes A_2$.

5.1 Pattern of a strongly connected component

In this paper, we suppose that a turn of any cycle may produce (or consume) **at most** one occurrence of a given action. In other terms, $-1 \leq \#(\mu,a) \leq 1$ for any action a of any cycle μ of $A_1 \otimes A_2$.

Consider a non-trivial *scc* C of a **free of deadlock** coverability product. It is worth noting that, by construction, the set of unbounded actions is the same for any node of C; we use $\Omega(C)$ to denote such a set. The relationships between unbounded actions within C consists to partition set $\Omega(C)$ to $\mathcal{P}(C) = \{P_1, \ldots, P_n\}$ such that for any node v of C, any cycle $v \xrightarrow{\mu} v$ and any part $P_j \in \mathcal{P}(C)$, the accumulated actions of $\Omega(C)$ verify the following equality: $a, b \in P_j \Leftrightarrow \#(\mu,a) = \#(\mu, b)$. $\mathcal{P}(C)$ is called the pattern of C.

We inductively build $\mathcal{P}(C)$, by considering the **elementary** cycles of C. Initially, the partition associated with C is empty. At step i, a new partition is computed according to the partition in the previous step $i-1$ by considering a new *elementary* cycle μ of C. Cycle μ may alter the relationships between some actions. In other terms, it may change the relationships between the part's elements of the partition computed in step $i-1$, since it brings some actions and consumes other actions, with the **same rhythm**. Thus, taking into account the restriction presented at the beginning of this section, each part P of the old partition is split into three subsets E, R and N.

- A subset E gathering actions of P produced by μ. Each action of E wins one occurrence for each turn of μ.
- A subset R gathering actions of P consumed by μ. Each action of R loses one occurrence for each turn of μ.
- A subset N gathering actions of P which are not changed by μ.

Furthermore, the set $E \subset \Psi(\mu, !)$ (resp. $R \subset \Psi(\mu, ?)$) whose actions don't belong to any part of the old partition is added to the partition being computed. E (resp. R) constitutes a fresh equality relationship.

Example 2 *Consider a partition* $\mathcal{P}_i(C) = \{\{a,b,c,d,e\},\{f,g\}\}$, *obtained at the* i^{th} *iteration, and a new elementary cycle* $\mu = !a\ !b\ ?b\ ?c\ !f\ ?e\ !h\ ?i?j$, *so:*
- $\Psi(\mu,!) = \{a,f,h\}$, $\Psi(\mu,?) = \{c,e,i,j\}$,

Algorithm 3: Pattern computation

Data: a strongly connected components C of $A_1 \otimes A_2$
Result: Pattern of C

```
partition ={∅};
foreach Elementary cycle μ of C do
    NewPartition=∅;
    foreach Part P of partition do
        E = {a ∈ P | #(μ,a) = 1};
        R = {a ∈ P | #(μ,a) = −1};
        N = {a ∈ P | #(μ,a) = 0};
        NewPartition=NewPartition ∪ {E, R, N};
    end
    E = {a ∈ Ψ(μ,!) | ∄P ∈ partition ∧ a ∈ P};
    R = {a ∈ Ψ(μ,?) | ∄P ∈ partition ∧ a ∈ P};
    NewPartition=NewPartition ∪ {E, R};
    partition=NewPartition;
end
return partition;
```

- Partition obtained at the next iteration $\mathcal{P}_{i+1}(C) = \{\{a\}, \{b,d\}, \{c,e\}, \{f\}, \{g\}, \{h\}, \{i,j\}\}$.

Part $\{a,b,c,d,e\}$ *is split into three subsets,* $\{a\}, \{b,d\}, \{c,e\}$ *since* $\#(\mu,a) = 1$, $\#(\mu,b) = \#(\mu,d) = 0$ *and* $\#(\mu,c) = \#(\mu,e) = -1$, *idem for the part* $\{f,g\}$. *Cycle* μ *brings new actions,* $h \in \Psi(\mu,!)$ *and* $i, j \in \Psi(\mu,?)$, *inducing two parts* $\{h\}$, $\{i,j\}$ *in* $\mathcal{P}_{i+1}(C)$. *It is worth noting that cycle* μ *has broken the equality of* a *with the other actions of its part in* $\mathcal{P}_i(C)$, *keeps an equality between* b *and* d *(resp.* c *and* e*) and so on.*

5.2 Choreography Checking

The pattern of a scc must C also take into account the elementary cycles of the ascendants scc of C. It is worth noting that the action occurrences, brought by elementary paths, are explicitly represented in the coverability product and are not included in the computation of the patterns of the scc. Consider again the coverability product of Figure 2, the pattern of scc $C0$ (resp. $C1$), computed in isolation is $\{\{c, a\}\}$ (resp. $\{\{c, a\},\{b\}\}$). The pattern of $C0$ is preserved when considering the ascendant scc, idem for $C1$.

Definition 7 *There is a choreography compatibility within a scc* C *of a coverability product if for any elementary cycle* μ *of* C, $\Psi(\mu,?) \in \mathcal{P}(C)$.

Consider an element $E \in \mathcal{P}(C)$. By construction, there is an equality relationship, in some nodes of $A_1 \otimes A_2$, between occurrences of actions of E. Any elementary cycle μ such that $\Psi(\mu,?) = E$ consumes, at each turn, a same number of elements of E preserving the relationship. Thus, the comsumption is *in a step with* the production rythm.

Example 3 *Consider the coverability graph of Figure 2. The pattern of its unique terminal Scc is* $\mathcal{P}(C1) = \{\{c,a\},\{b\}\}$. *There is a choreography compatibility within* $C1$, *since cycles of* $C1$ *fulfill definition 7. For instance,* $\Psi(\mu,?) = \{a,c\} \in \mathcal{P}(C1)$ *for cycle* $\mu = ?c\ !b\ ?a \in C1$, *idem for* $\Psi(\mu,?) = \{b\} \in \mathcal{P}(C1)$ *for cycle* $\mu = ?b \in C1$.

5.3 UR compatibility Checking

The UR compatibility requires that each message sent is eventually received. To ensure that all sent messages are received, we must have in all terminal strongly connected components, cycles able to consume accumulated actions. These cycles are garbage collectors.

Definition 8 *A cycle* μ *of* $A_1 \otimes A_2$ *is a garbage collector iff:* $\Psi(\mu,?) \neq \emptyset$ *and* $\Psi(\mu,!) = \emptyset$.

A garbage collector is able to clean the actions $\Psi(\mu,?)$. In the other hand, the cycle does not produce residual messages. Consider again the coverability product of Figure 2. The cycle labelled by $?b$ is a garbage collector of action b, whereas the cycle labelled by $?c!b?b?a$ is a garbage collector of actions a and b.

Proposition 1 *Two automata* A_1 *and* A_2 *are UR-compatible if the coverability product* $A_1 \otimes A_2$ *is free of deadlock and for any terminal scc* C *of* $A_1 \otimes A_2$ *the following conditions hold:*

- *There is a choreography compatibility within* C.
- *For any node* v *of* C *and any action* a *of* $\Omega(v)$, *there is a garbage collector of* a.
- $\forall a \in \Sigma$, $\exists$ *a node* v *of* C, *such that* $M(v)(a) = 0$.

Proof. The first point states that there are proper interactions between peers, while the second ensures that peers are able to clean the accumulated actions. Finally, the last point indicates that the peer's buffers always reach a state where any action brought by an elementary path has been emptied.

The coverability product of Figure 2 holds the conditions of the previous proposition. Thus, the peers of example 1 are UR-compatible.

6 Conclusion

In this paper, we presented a new approach to check compatibility of asynchronous communicating infinite systems, using unbounded and unordered buffers. We do not have any restrictions on the number of cycle iterations, size of buffers nor on number of components. Our main result is that our approach does not require any synchronizability property. In our approach, we proposed a coverability product by constructing a *finite* state space. We define the concept of

patterns associated with messages occurring in the cycles of a strongly connected component. These patterns are jointly used with the covering graph to check the UR-compatibility of the components. The patterns allow to make an underlying analysis of strongly connected components. Thus, they show if there is a good choreography between components.

We have implemented our approach, which is under experimentation as future work, we aim to consider systems with FIFO unbounded buffers. This work could be a promising base to tackle with infinite systems.

References

1. L. Alfaro and T.A. Henzinger. Interface automata. In *Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering (FSE), ACM*, pages 109–120. Press, 2001.
2. Samik Basu and Tevfik Bultan. On deciding synchronizability for asynchronously communicating systems. *Theor. Comput. Sci.*, 656:60–75, 2016.
3. Samik Basu, Tevfik Bultan, and Meriem Ouederni. Deciding choreography realizability. In *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*, pages 191–202, 2012.
4. Michael Blondin, Alain Finkel, Christoph Haase, and Serge Haddad. Approaching the coverability problem continuously. In *TACAS 2016, Held as Part ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, pages 480–496, 2016.
5. Ahmed Bouajjani and Michael Emmi. Bounded phase analysis of message-passing programs. *STTT*, 16(2):127–146, 2014.
6. Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, April 1983.
7. C. Canal, P. Poizat, and G. Salaun. Model-based adaptation of behavioral mismatching components. *IEEE Transactions on Software Engineering*, 34(4):546–563, 2008.
8. Djaouida Dahmani, Mohand Cherif Boukala, and Hassen Mountassir. A Petri net approach for reusing and adapting components with atomic and non-atomic synchronisation. In *International Workshop on Petri Nets and Software*, Tunis, Tunisia, Juin 2014.
9. Djaouida Dahmani, Mohand Cherif Boukala, and Hassen Mountassir. Reusing and adapting components using atomic and non-atomic strong synchronisations. In *Conférence francophone sur l'Architecture Logicielle, CAL'2014*, Paris, France, 10-11 Juin 2014.
10. Serge Haddad, Rolf Hennicker, and Mikael H. Møller. Channel properties of asynchronously composed petri nets. In *Application and Theory of Petri Nets and Concurrency - 34th International Conference, PETRI NETS 2013, Milan, Italy, June 24-28, 2013. Proceedings*, pages 369–388, 2013.
11. Rolf Hennicker, Michel Bidoit, and Thanh-Son Dang. On synchronous and asynchronous compatibility of communicating components. In *Coordination Models and Languages - 18th IFIP WG 6.1 International Conference, COORDINATION 2016, Held as Part of the 11th International Federated Conference on Distributed Computing Techniques, DisCoTec 2016, Heraklion, Crete, Greece, June 6-9, 2016, Proceedings*, pages 138–156, 2016.

12. Olivia Oanea and Karsten Wolf. An efficient necessary condition for compatibility. In Oliver Kopp and Niels Lohmann, editors, *ZEUS*, volume 438 of *CEUR Workshop Proceedings*, pages 81–87. CEUR-WS.org, 2009.
13. Meriem Ouederni, Gwen Salaün, and Tevfik Bultan. Compatibility checking for asynchronously communicating software. In *Formal Aspects of Component Software - 10th International Symposium, FACS 2013, Nanchang, China, October 27-29, 2013, Revised Selected Papers*, pages 310–328, 2013.

Complexity Aspects of Web Services Composition

Karima Ennaoui, Lhouari Nourine, and Farouk Toumani

LIMOS, CNRS, Universite Clermont Auvergne
1 Rue de la Chebarde, 63178 AUBIERE CEDEX

Abstract. The web service composition problem can be stated as follows: given a finite state machine M, representing a service business protocol, and a set of finite state machines $\mathcal{R}$, representing the business protocols of existing services, the question is to check whether there is a simulation relation between M and the shuffle product closure of $\mathcal{R}$. In fact the shuffle product is a subclass of the communication free petri net and basic parallel processes, for which the same problem of simulation is known to be 2-Exptime-hard.
This paper studies the impact of several parameters on the complexity of this problem. We show that the problem is *Exptime-complete* if we bound either: *(i)* the number of instances of services in $\mathcal{R}$ that can be used in a composition, or*(ii)* the number of instances of services in $\mathcal{R}$ that can be used in parallel, or *(iii)* the number of the so-called hybrid states in the finite state machines of $\mathcal{R}$ by 2.

1 Introduction

Web Services [2] is a new computing paradigm that tends to become a technology of choice to facilitate interoperation among autonomous and distributed applications. The UDDI consortium defines Web services as *self-contained, modular business applications that have open, Internet-oriented, standards-based interfaces*. Several models have been proposed in the literature to describe different facets of services. In particular, the importance of specifying external behaviour of services, also called service business protocols, has been highlighted in several research works [6,3,4]. Through literature, different models have been used to represent web service business protocols. The Finite States Machines (FSM) formalism is widely adopted in this context to model *statefull* applications exposed as web services where states represent the different phases that a service may go through while transitions represent "abstract" activities that a service can perform [6,3,4].

We consider in this paper the problem of Web Service Composition (WSC). This problem arises from the situation where none of the existing services can provide a requested functionality. In this case, the idea is to find out, algorithmically, if the target functionality could be composed out of the existing services (components repository). This automatic approach of composition simplifies the development of software by reusing existing components and offers capabilities to customize complex systems built on the fly [9]. We focus more particularly on

a specific instance of WSC, namely the (business) protocol synthesis problem, which can be stated as follows: given a set of business protocols of available services and given a business protocol of a target service, is it possible to synthesize automatically a mediator that *implements* the target service using the existing ones?

[16] shows that when business protocols are described by means of FSMs, the WSC problem can then be formalized as the problem of deciding whether there exists a simulation relation between the target protocol and the shuffle (or asynchronous product) of the available ones. This result is however based on the implicit assumption that at most one *instance* of each available service can be used in a composition. This setting has been extended in [9] to the case where the number of instances that can be used in a composition is unbounded. WSC is formalized in this latter case as a simulation problem between an FSM and an infinite state machine, called Product Closure State Machine (PCSM), that is able to compute the shuffle closure of an FSM.

Shuffle product of FSMs (and PCSM) is a subclass of Basic Parallel Processes (BPP), the class of communication free petri nets: every transition has at most one input place. Simulation of FSM by BPP was proven Expspace-hard by Lasota [15] and 2-Exptime-hard in [8].

Complexity analysis of WSC was first considered by Musholl *et al.*[16], under the aforementioned implicit assumption, where it is shown Exptime-Complete. In case of unbounded instances, the WSC problem has been proved decidable with an Ackermanian function as upper bound in [9]. The proof of [9] is based on Dickson lemma, and hence cannot be exploited to derive tighter upper bounds. An Expspace-hard lower bound is given by Lasota[15]. The source of complexity derived from the analysis of the algorithm given in [9] is related to the presence of the so-called hybrid states (final states with outgoing transitions and correspond to unbounded places in Petri net terminology) in the components and loops in the target: if the target FSM is loop free, the WSC problem becomes NP-complete and when the components are hybrid state free the problem is proven Exptime.

In this paper, we consider additional parameters related to bounded/unbounded web services composition. We consider as inputs an FSM M (the target protocol) and a set of FSMs $\mathcal{R}$ (the protocols of the available services) and we investigate the complexity of testing simulation between M and the shuffle closure of $\mathcal{R}$, represented as a PCSM [9]. More precisely, we study the complexity of the following problems:

1. $WSC(M, \mathcal{R})$: The problem of composing M using an unbounded number of instances of $\mathcal{R}$.
2. $BC(M, \mathcal{R}, k)$: The problem of composing M using at most k instances of each FSM in $\mathcal{R}$.
3. $PBC(M, \mathcal{R}, k)$: The problem of composing M using simultaneously at most k FSM instances in $\mathcal{R}$ (in parallel).
4. $UCHS(M, \mathcal{R}, k)$: The problem of composing M using an unbounded number of instances of $\mathcal{R}$, with the number of hybrid states in $\mathcal{R}$ is bounded by $k \in \{0, 1, 2\}$.

Table in figure 1 displays known and new complexity results regarding the WSC problem.

M	Acyclic FSM	general FSM
$BC(M, \mathcal{R}, 1)$	NP-complete[9]	Exptime-complete [16]
$BC(M, \mathcal{R}, k)$	NP-complete[9]	Exptime-complete
$PBC(M, \mathcal{R}, 1)$	Polynomial	Polynomial
$PBC(M, \mathcal{R}, k)$	NP-complete	Exptime-complete
$WSC(M, \mathcal{R})$	NP-complete [9]	Decidable [9]
$UCHS(M, \mathcal{R}, 0)$	NP-complete[9]	Exptime-complete
$UCHS(M, \mathcal{R}, 1)$	NP-complete[9]	Exptime-complete
$UCHS(M, \mathcal{R}, 2)$	NP-complete[9]	Exptime-complete

Fig. 1. Complexity results of WSC sub-problems

Paper organisation Section 2 recalls some basic definitions needed in this paper. In section 3, we investigate the problem of bounded web services composition and proves that it is Exptime-Complete. Next, we define web service composition with fixed number of parallel instances, and show that it is Exptime-Complete in general and is NP-complete when M is loop free and polynomial for $k = 1$. In section 5, we consider the web service composition when the number of hybrid states is bounded. We show that this problem is Exptime-Complete for $k = 0$, $k = 1$ and $k = 2$. We conclude in section 6.

2 Preliminaries

Finite State Machine We consider in this paper service business protocols formally described as FSMs. We recall below the definition of such machines.

Definition 1. ***(Finite State Machine (FSM))***
A State Machine ***(SM)*** *M is a tuple $M = (\Sigma_M, Q_M, F_M, q_M^0, \delta_M)$, where: Σ_M is a finite alphabet, Q_M is a set of states, $\delta_M \subseteq Q_M \times \Sigma_M \times Q_M$ is a set of labelled transitions, $F_M \subseteq Q_M$ is a set of final states, and $q_M^0 \in Q_M$ is the initial state. If Q_M is finite then M is called a Finite State Machine (FSM).*

Moreover, a state $q \in Q_M$ is called: **intermediate**, if $q \notin F_M$ and $\exists p_1, p_2 \in Q_M$, $s.t$ $(p_1, a, q) \in \delta_M$ and $(q, b, p_2) \in \delta_M$, we denote by $I(M)$ the set of intermediate states of M; **hybrid**, if $q \in F_M$, $q \neq q_0$ and there exist at least one transition $(q, b, p) \in \delta_M$, with $p \in Q_M$ and $b \in \Sigma$, the set of hybrid states is denoted $H(M)$ and **terminal**, if $q \in F_M$ and is not hybrid.

We define the **norm of a state** q as the finite length of the shortest path from q to a final state. **The norm of an FSM** M, noted *norm(M)*, is the maximal norm of its states.

k-Iterated Product Machine (k-IPM) and Product State Machine (PCSM) We start by defining the shuffle (asynchronous product) and union operations on FSMs:

Definition 2. ***(Asynchronous product and Union of two FSMs)*** *Let $M = (\Sigma_M, Q_M, F_M, q_M^0, \delta_M)$ and $M' = (\Sigma_{M'}, Q_{M'}, F_{M'}, q_{M'}^0, \delta_{M'})$ be two FSMs. We have :*

- *The* ***shuffle or asynchronous product*** *of M and M', denoted $M \times M'$, is an FSM $(\Sigma_M \cup \Sigma_{M'}, Q_M \times Q_{M'}, F_M \times F_{M'}, (q_M^0, q_{M'}^0), \lambda)$ where the transition function λ is defined as follows: $\lambda = \{((q, q'), a, (q_1, q_1{}')) : ((q, a, q_1) \in \delta_M$ and $q' = q_1')$ or $((q', a, q_1{}') \in \delta_{M'}$ and $q = q_1)\}$.*
- *The* ***union*** *of M and M', denoted $M \cup M'$, is the FSM $(\Sigma_M \cup \Sigma_{M'} \cup \{\epsilon\}$, $Q_M \cup Q_{M'} \cup \{q_0\}$, $F_M \cup F_{M'}$, q_0, $\delta_M \cup \delta_{M'} \cup \{(q_0, \epsilon, q_M^0), (q_0, \epsilon, q_{M'}^0)\})$.*

For a set of available FSMs $\mathcal{R} = \{M_1, ...M_i\}$, we consider a compact structure that abstracts all possible executions that can be produced using the components of $\mathcal{R}$. First, we begin by the simple case where each M_j can be used only once:

Definition 3. ***(Union of asynchronous products of FSMs set)*** *Let $\mathcal{R} = \{M_1....M_m\}$ be a FSMs repository. We define $\odot(\mathcal{R})$ the union of asynchronous product of all the subsets of $\mathcal{R}$ as the FSM: $\odot(\mathcal{R}) = \bigcup_{\{M_{i_1},...,M_{i_j}\} \in 2^{\mathcal{R}}} (M_{i_1} \times ... \times M_{i_j})$ where $j \in [0, i]$.*

Second, we consider the case where the number of copies of each $M_j \in \mathcal{R}$ is bounded by an integer k:

Definition 4. ***(k-iterated product of FSMs set $\mathcal{R}$)*** *The* ***k-iterated product of*** *$\mathcal{R}$ is defined by $\mathcal{R}^{\otimes_k} = \mathcal{R}^{\otimes_{k-1}} \times \odot(\mathcal{R})$ with $\mathcal{R}^{\otimes_1} = \odot(\mathcal{R})$.*

Finally, we consider the general case where the number of instances of each $M_j \in \mathcal{R}$ is unbounded. This corresponds to the product closure of $\mathcal{R}$ [9]:

Definition 5. ***(Product closure of FSMs set)*** *The* ***product closure*** *of $\mathcal{R}$, noted $\mathcal{R}^{\otimes}$, is defined as: $\mathcal{R}^{\otimes} = \bigcup_{i=0}^{+\infty} \mathcal{R}^{\otimes_i}$.*

The **P**roduct **C**losure **M**achine (**PCSM**) of $\mathcal{R}$, defined in [9] and proven equivalent to $\mathcal{R}^{\otimes}$, is the SM with unbounded number of tokens stacked at the beginning in the initial states in $\mathcal{R}$. Then, the instantaneous description of a PCSM gives the number of tokens (instances) at each state of its underlaying FSMs. This description is called a configuration of $\mathcal{R}^{\otimes}$. We omit from this description the initial states (source:infinite number of tokens) and terminal states (sink:terminated instances).

Example 1. Figure 2 illustrates the execution of the sequence "*abca*" by the PCSM of the FSM M in figure 2-(a). M contains one intermediate state q_1 and two hybrid states q_2 and q_5. Therefore, figure 2-(b) depicts a part of the PCSM $M^{\otimes}$ with triplets as configurations where integers witness respectively

the number of tokens in q_1, q_2 and q_5. For each configuration c in figure 2-(b), we associate an instant t (or several instants) during the execution when c describes the PCSM. At the beginning ($t = 0$), $M^{\otimes}$'s instantaneous description is $(0, 0, 0)$, interpreting an empty stack in every state of M, except the initial state with an infinite number of tokens (figure 2-(c)). To execute the transition (q_0, a, q_1), a token is moved from q_0 to q_1 in figure 2-(d), corresponding to the configuration $(1, 0, 0)$ in instant $t = 1$. In $t = 2$, the executed transition (q_0, b, q_4) corresponds to moving a token from the initial state to a terminal one q_4 (figure 2-(e)). Since the instantaneous description does not consider neither initial states nor terminal ones, then the configuration stays the same as the previous instant. Notice that this move corresponds to both creating and terminating an instance of the FSM. Then, the transition (q_0, c, q_5) is executed by moving a token from q_0 to the hybrid state q_5. This creates a new instance implying, in this case, an increase in the number of simultaneously used instances in the execution. This is depicted in figure 2-(f). Finally, a token is moved from the state q_1 to q_2 in figure 2-(g), in order to execute the transition (q_1, a, q_2). It changes $M^{\otimes}$'s instantaneous description in $t = 4$ into $(0, 1, 1)$ which is a final configuration (i.e $(0, 1, 1) \in F_C$) since all tokens in the PCSM are in final states (either hybrid or terminal).

Formally, we define the PCSM $\mathcal{R}$ as the SM $(\Sigma_{\mathcal{R}}, \mathcal{C}_{\mathcal{R}^{\otimes}}, F_{\mathcal{C}}, c_0, \Phi_{\mathcal{R}^{\otimes}})$, where:

1. $\Sigma_{\mathcal{R}} = \bigcup_{M_j \in \mathcal{R}} \Sigma_{M_j}$;
2. $\mathcal{C}_{\mathcal{R}^{\otimes}}$ is the set of states (also called configurations of $\mathcal{R}^{\otimes}$). $\mathcal{C}_{\mathcal{R}^{\otimes}} \subset \mathbb{N}^n$, with: $n = n_I(\mathcal{R}) + n_H(\mathcal{R})$ with: $n_I(\mathcal{R})$ is the number of intermediate states in $\mathcal{R}$) ($n_I(\mathcal{R}) = |\bigcup_{M_j \in \mathcal{R}} I(M_j)|$) and $n_H(\mathcal{R})$ is the number of hybrid states in $\mathcal{R}$) ($n_H(\mathcal{R}) = |\bigcup_{M_j \in \mathcal{R}} H(M_j)|$). For each configuration c, $c[m]$ (the mth component of c) is called a witness of a unique state $q_m \in Q_{M_j}$ for some j. Note that:
   - q_m is an intermediate state, if $1 \leq m \leq n_I(\mathcal{R})$;
   - q_m is an hybrid state, if $n_I(\mathcal{R}) + 1 \leq m \leq n$.

   In an abuse of notation, we use $c[m]$ and $c[q_m]$ interchangeably.
3. $F_{\mathcal{C}}$ is the set of final states. $F_{\mathcal{C}} = \{c \in \mathcal{C}_{\mathcal{R}^{\otimes}} | c[m] = 0,$ *for each:* $1 \leq m \leq n_I(\mathcal{R})\}$;
4. $c_0 = \{0\}^n$ is the initial state of $\mathcal{R}^{\otimes}$;
5. $\Phi_{\mathcal{R}^{\otimes}} \subseteq \mathcal{C}_{\mathcal{R}^{\otimes}} \times \Sigma_{\mathcal{R}} \times \mathcal{C}_{\mathcal{R}^{\otimes}}$ is the set of transitions. we have $(c_1, a, c_2) \in \Phi_{\mathcal{R}^{\otimes}}$ **iff**:
   - there exists $(q_0, a, q) \in Q_{M_j}$, such that: q_0 is the initial state of M_j and $c_2[q] = c_1[q] + 1$ and $c_2[p'] = c_1[p']$ for each $p' \neq q$.
   - there exists $(p, a, q) \in Q_{M_j}$, such that: $c_2[p] = c_1[p] - 1$, $c_2[q] = c_1[q] + 1$ and $c_2[p'] = c_1[p']$ for each $p' \neq p, q$.
   - there exists $(p, a, q) \in Q_{M_j}$, such that: q is a final state or the initial state, $c_2[p] = c_1[p] - 1$ and $c_2[p'] = c_1[p']$ for each $p' \neq p$.

Simulation preorder We recall below the definition of the simulation preorder between two SMs.

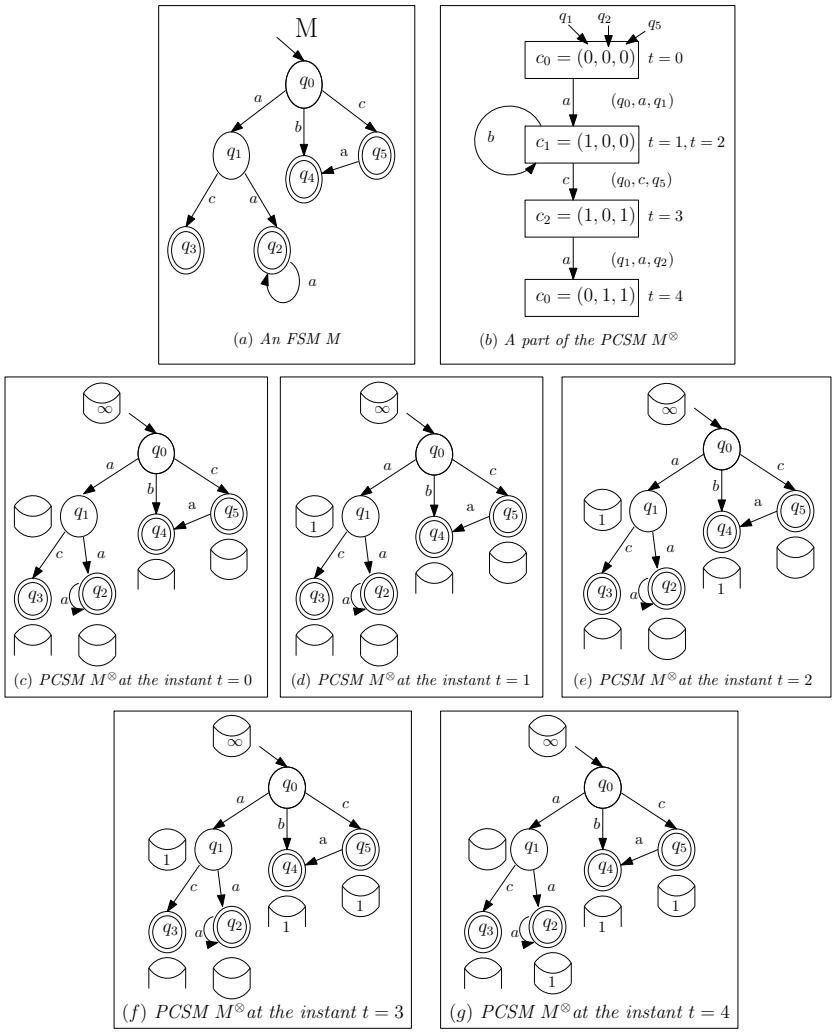


Fig. 2. An example of execution of a sequence using a PCSM.

Definition 6. ***(Simulation)***
Let $M = (\Sigma_M, Q_M, F_M, q_M^0, \delta_M)$ and $N = (\Sigma_N, Q_N, F_N, q_N^0, \delta_N)$ be two SMs. A state $p \in Q_M$ is simulated by a state $q \in Q_N$, denoted $p \ll_{(M,N)} q$ ($p \ll q$ when M and N are understood from context), iff the following two conditions hold:

1. *$\forall a \in \Sigma_M$ and $\forall p' \in Q_M$ such that $(p, a, p') \in \delta_M$, there exists $(q, a, q') \in \delta_N$ such that $p' \ll q'$, and*
2. *if $p \in F_M$, then $q \in F_N$.*

M is simulated by N, denoted $M \ll N$, ***iff*** *the initial state of N simulates the initial sate of M.*

Example 2. Figure 3-(c) is an example of a simulation tree, verifying if the initial state s_0 of the FSM A (figure 3-(a)) is simulated by the initial configuration $c_0 = (0, 0, 0)$ of the PCSM of M (figure 3-(b)). A branch is terminated with success when a terminal state of A is reached and paired with a final configuration (all intermidiate witnesses are null), or when a configuration of $M^{\otimes}$ that covers one of its predecessors is reached and paired with the same state of A. In this case, the simulation tree proves that $A \ll M^{\otimes}$.

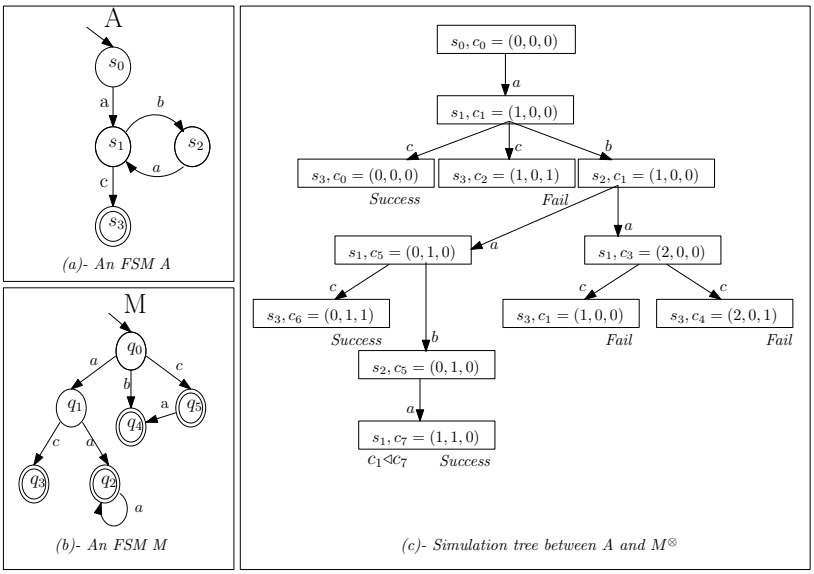


Fig. 3. An example of a simulation tree.

Interestingly, the simulation verification defined above can be seen as a two players game in a directed graph (V_a, V_d, δ, v_0), such that $V = V_a \cup V_d$ is the set of vertices with $V_a \subseteq Q_M \times Q_N$ and $V_d \subseteq Q_M \times Q_N \times \Sigma_M$, $\delta \subseteq (V_a \times V_d) \cup (V_d \times V_a)$ is the edges set verifying:
-for $(q, p) \in V_a$ and $(q, a, q') \in \delta_M$, we have $((q, p), (q', p, a)) \in \delta$; and
-for $(q, p, a) \in V_d$ and $(p, a, p') \in \delta_N$, we have $((q, p, a), (q, p')) \in \delta$.

The game is played by an attacker and a defender. It starts by putting a token in $v_0 = (q_M^0, q_N^0) \in V_a$, then the players move it along the edges of the graph. If the token is on a vertex $v \in V_a$ then the attacker moves it, otherwise it is the defender's turn.

A strategy of a player $x \in \{a, d\}$ is a function $S : V^*.V_x \mapsto V$, where $V^*.V_x$ denotes all sequences of vertices in V that end with a vertex in V_x and

$S(v_0, ..., v_k) = v_{k+1}$ implies that $(v_k, v_{k+1}) \in \delta$. In each different play, a player x adapts a strategy that decides his moves.

The defender wins every infinite play. Otherwise, the first player who can not move loses. M is simulated by N **iff** the defender has a wining strategy regardless of his opponent's strategy.

Observe that, by definition, each transition of a PCSM can at most increase or decrease a configuration component by 1. In addition, if a configuration is final then all intermediate states witnesses are equal to 0. Therefore, given a set of FSMs $\mathcal{R}$ and $c \in \mathcal{C}_{\mathcal{R}^\otimes}$, we have $\Sigma_{q \in \bigcup_{M_i \in \mathcal{R}} I(M_i)} c[q] \leq norm(c)$. Moreover, since final states can only be simulated by final ones, then for M an FSM and $p \in Q_M$, if $p \ll c$ then $norm(c) \leq norm(p)$. Hence, we are able to derive the following property.

Property 1. **(Intermediate witnesses bound) [9]** For $c \in \mathcal{C}_{\mathcal{R}^\otimes}$ and $p \in Q_M$, if $p \ll c$ then $\Sigma_{q \in \bigcup_{M_i \in \mathcal{R}} I(M_i)} c[q] \leq norm(p)$. We denote $\mathcal{C}^M_{\mathcal{R}^\otimes} = \{c \in \mathcal{C}_{\mathcal{R}^\otimes} | \Sigma_{q \in \bigcup_{M_i \in \mathcal{R}} I(M_i)} c[q] \leq norm(M)\}$.

In [9], the WSC problem in the unbounded case is reduced to simulation test between an FSM and a PCSM and this later problem is proved to be decidable. The proof of the termination of the algorithm given in [9] is based on the following property:

Property 2. **(configuration cover) [9]** Let c and c' be two configurations of $\mathcal{R}^\otimes$, such that: $c[m] = c'[m], m \in [1, n_I(\mathcal{R})]$ *and* $c[m] \leq c'[m], m \in [n_I(\mathcal{R})+1, n]$. if $q \ll c$, where q is a state of a SM M, then $q \ll c'$.

We say that c' covers c, denoted $c \lhd c'$.

We introduce below the algorithm of [9], focusing the presentation on the structure of its execution tree.

Definition 7. ***(Simulation Tree)***
We call a simulation tree $T_{sim}(M, \mathcal{R}^\otimes)$ a tree (V, v_0, E) where:

- $v_0 = (q^0_M, c_0)$ *is the root of the tree;*
- $V \subset Q_M \times \mathcal{C}^M_{\mathcal{R}^\otimes}$ *is the set of nodes;*
- *If* $(q, c) \in V$ *and* q *is final in* M *then so is* c *in* $\mathcal{R}^\otimes$;
- $E \subset V \times V$ *is the set of the tree's edges.* $\forall e = ((p, c), (q, d)) \in E : \exists a \in \Sigma_M$ *s.t* $(p, a, q) \in \delta_M$ *and* $(c, a, d) \in \Phi_{\mathcal{R}^\otimes}$*;*
- $v = (p, c) \in V$ *is a leaf in* $T_{sim}(M, \mathcal{R}^\otimes)$ *iff* p *is terminal in* A *or there exists an ancestor* $(p, c') \in V$ *of* v *in* $T_{sim}(M, \mathcal{R}^\otimes)$ *such that* $c \lhd c'$.

In the next section, we shall bound the size of this tree in the case of bounded WSC problem (i.e., when the instances of services allowed to be used in the simulation is bounded by a parameter k).

3 Bounded Composition

We call a *bounded* WSC problem, a service composition problem where the number of copies of each web service in the repository $\mathcal{R}$ used to compose the target M is bounded a priori by an integer k. This problem is formally stated as follows.

Problem 1. **Bounded Composition** $BC(M, \mathcal{R}, k)$
Input : $\mathcal{R}$ a set of FSMs; M a target FSM; k an integer.
Question : $M \ll \bigcup_{i=0}^{k} \mathcal{R}^{\otimes_i}$?

The particular case $BC(M, \mathcal{R}, 1)$ has been investigated by Muscholl and Walukiewicz [16] where it is shown to be Exptime-Complete. We shall prove in this section that $BC(M, \mathcal{R}, k)$ is also Exptime-Complete. We point out that the straightforward reduction of $BC(M, \mathcal{R}, k)$ to $BC(M, \mathcal{R}, 1)$, obtained by duplicating k times each service of $\mathcal{R}$, is not polynomial in the input size, since k may be large, and hence cannot be used to achieve our goal.

The parameter k drops the infinite aspect and reduces the search space. In this case, a loop in M can only be simulated by loops in $\mathcal{R}$. For example, one can observe that, in figure 4, S_t is not simulated by $\bigcup_{i=0}^{k}\{R_1, R_3\}^{\otimes_i}$ for every $k \in \mathbb{N}$. This is because when we repeat the loop in S_t $(k+1)$ times, there is no corresponding execution in $\bigcup_{i=0}^{k}\{R_1, R_3\}^{\otimes_i}$. However, we have $S_t \ll \bigcup_{i=0}^{k}\{R_1, R_3\}^{\otimes_i}$, for any $k \geq 1$.

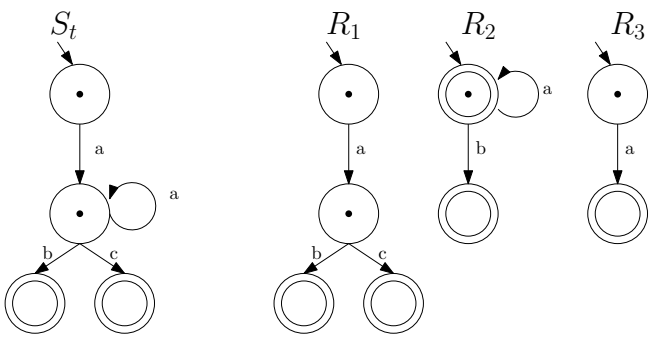


Fig. 4. A yes instance of $BC(M, \mathcal{R}, k)$ with $k = 1$.

In the following, we give an upper bound of the number of states that might appear in $\bigcup_{i=0}^{k} \mathcal{R}^{\otimes_i}$, with $k \in \mathbb{N}$.

Lemma 1. *Let $\mathcal{R}$ be a set of FSM and k is an integer. The number of states in $\bigcup_{i=0}^{k} \mathcal{R}^{\otimes_i}$ is bounded by $O(2^{nlogk})$ where $n = n_I(\mathcal{R}) + n_H(\mathcal{R})$.*

Proof. Notice that $\mathcal{R}^{\otimes} = (\bigcup_{i=0}^{k} \mathcal{R}^{\otimes_i}) \cup (\bigcup_{i=k+1}^{+\infty} \mathcal{R}^{\otimes_i})$.

In fact, the states in $\bigcup_{i=0}^{k} \mathcal{R}^{\otimes_i}$ correspond to the PCSM's configurations subset $\{c \in \mathcal{C}_{\mathcal{R}^{\otimes_k}} \mid 0 \leq c[i] \leq k, i \in [1, n]\}$. Hence, the number of states of $\bigcup_{i=0}^{k} \mathcal{R}^{\otimes_i}$ is bounded by $(k+1) \times \ldots \times (k+1) = 2^{nlog(k+1)}$. □

This lemma reduces the search space to an exponential size and leads to the following theorem.

Theorem 1. *$BC(M, \mathcal{R}, k)$ is Exptime-Complete*

Proof. `Exptime.` To show that $BC(M, \mathcal{R}, k)$ is Exptime, we bound the size of the simulation tree. A node of the simulation tree corresponds to (q, c) where q is a state of M and c a configuration of $\mathcal{R}^{\otimes_k}$. According to Lemma 1, the number of PCSM's configurations is bounded by k^n. So the number of nodes in the simulation tree is at most $|Q_M| \times k^n = 2^{nlog(k)+log(|Q_M|)}$ and therefore the complexity is in Exptime.

`Exptime-Hardness.` It can be deducted directly from the Exptime-Hardness of the particular case $BC(M, \mathcal{R}, 1)$ [16]. □

Instead of the total number of instances used in the simulation, what happens if we bound only the number of instances used simultaneously? we raise this question in the next section and prove that the problem stays Exptime-complete.

4 Bounded parallel instances

Now we consider a new parameter in service web composition that bounds the number of communications in parallel between the target and the services, i.e. the number of live services executions is bounded, but the number of instances is not. It appears that the web services composition with unbounded instances and bounded parallel instances is Exptime-Complete.

To do so, we limit the configurations of the PCSM $\mathcal{R}^{\otimes}$ to configurations where the number of waiting instances is bounded by k. Indeed, when we need to use a new instance in $\Phi_{\mathcal{R}^{\otimes}}$, we check if $\sum_{i=1}^{n} c[i] \geq k$. If so, we decrease $c[j]$ for some $j \in [n_I(\mathcal{R}) + 1, n_H(\mathcal{R})]$, i.e. we finish an instance that is waiting in an hybrid state. Let us denote by $\mathcal{R}^{\otimes_{k,p}}$ the obtained PCSM.

Problem 2. **Bounded Parallel Instances Composition ($PBC(M, \mathcal{R}, k)$)**
Input : $\mathcal{R}$ a set of FSMs;
M a target FSM.
k an integer, bounding the number of parallel instances of $\mathcal{R}$'s components used simultaneously in the simulation.
Question : $M \ll \mathcal{R}^{\otimes_{k,p}}$?

Note that $PBC(M, \mathcal{R}, k)$ can use an unbounded number of instances but only k instances in parallel.

Theorem 2. *$PBC(M, \mathcal{R}, k)$ is Exptime-complete.*

Proof. First we show that $PBC(M, \mathcal{R}, k)$ is Exptime. Clearly the entry of any configuration is bounded by k (hybrid states are included) and therefore we can check simulation in Exptime, since the depth of the simulation tree is bounded by k^n (see Lemma 1).

To show the Exptime-hardness, it suffices to note that the unbounded composition without hybrid states $UCHS(M, \mathcal{R}, 0)$ is a particular case of $PBC(M, \mathcal{R}, k)$, since we prove later in theorem 4 that $UCHS(M, \mathcal{R}, 0)$ is Exptime-hard. In fact, the number of tokens in intermediate states of $\mathcal{R}$ is bounded by $norm(M)$ (property 1). Hence, when $\mathcal{R}$ is hybrid state free, the number of instances that can be used in the simulation is bounded by $norm(M)$. In other words, it corresponds to $PBC(M, \mathcal{R}, norm(M))$. □

For k a constant, we obtain the following.

Corollary 1. *$PBC(M, \mathcal{R}, k)$ is polynomial when k is a constant.*

Proof. First of all, let us consider for every configuration c of $\mathcal{R}^{\otimes_{k,p}}$, a new component $c[n+1] = k - (\sum_{i=1}^{n} c[i])$, with $n = n_I(\mathcal{R}) + n_H(\mathcal{R})$.

For every configuration c in $\mathcal{R}^{\otimes_{k,p}}$, the non-empty witnesses $\{c[i] > 0, \leq i \leq n+1\}$ correspond to a partition of k elements (instances) into a sequence of j non empty subsets, for $j = |\{c[i] > 0, 1 \leq i \leq n\}| \leq k$. Note that j is in fact inferior to $min(k, n)$, but since k is a constant then it is more interesting to keep it as a lower bound of j.

For every $j \leq k$, the number of labeled partitions of k elements into a sequence of j non empty subsets is $j! \times \{{k \atop j}\}$, where $\{{k \atop j}\}$ is a Stirling number of the second kind [1]. Hence, the number of configurations in $\mathcal{R}^{\otimes_{k,p}}$ that have j non-empty witnesses is bounded by $C_n^j \times j! \times \{{k \atop j}\}$. Notice that $C_n^j = e\frac{n \ldots \times (n-j+1)}{j!}$ is in the order of $O(n^j)$.

We conclude that the number of configurations in $\mathcal{R}^{\otimes_{k,p}}$ is bounded by $\sum_{j=1}^{k} C_n^j \times j! \times \{{k \atop j}\} \in O(n^k)$.

Finally, by applying the simulation algorithm in [10], $PBC(M, \mathcal{R}, k)$ can be decided in $O(m_v.m_e)$, where $m_v = |Q_M| + |\mathcal{C}_{\mathcal{R}^{\otimes}}|$ and $m_e \leq |Q_M|^2 + |\mathcal{C}_{\mathcal{R}^{\otimes}}|^2$ are respectively the number of edges and transitions in M and $\mathcal{R}^{\otimes_{k,p}}$. □

In the following, we show that $PBC(M, \mathcal{R}, k)$ is NP-Complete for loop-free target FSM. Let μ a sequence of letters (a word) over Σ and M the FSM that recognizes exactly μ. We call $\mu^{\otimes}$ the language recognized by $M^{\otimes}$. We consider the following NP-complete Problem [13].

Problem 3. **SHUFFLE PRODUCT**

Input : μ and μ' two words over an alphabet Σ;
Question : $\mu \in \mu'^{\otimes}$?

Theorem 3. *$PBC(M, \mathcal{R}, k)$ is NP-complete whenever M is loop-free.*

Proof. Clearly $PBC(M, \mathcal{R}, k)$ is in NP since the simulation relation is polynomial in the size of M. To show the NP-hardness, we reduce SHUFFLE PRODUCT to it. Let μ and μ' be an instance of SHUFFLE PRODUCT. We associate

an FSM M which recognizes exactly μ and $\mathcal{R} = \{N\}$ where N is the FSM that recognizes exactly μ'. Since M is a chain, then the size of a branch of the simulation tree can not surpass $|\mu|$. Thus,the simulation verification will only explore $\mathcal{R}^{\otimes_{k,p}}$'s executions where the size is bounded by $|\mu| \leq k.|\mu'|$ with $k = \lceil \frac{|\mu|}{|\mu'|} \rceil$ and therefore the number of instances is bounded by k. Hence, $\mu \in \mu'^{\otimes}$ iff $M \ll \mathcal{R}^{\otimes_{k,p}}$ iff $M \ll \mathcal{R}^{\otimes}$. We give an example in figure 5. □

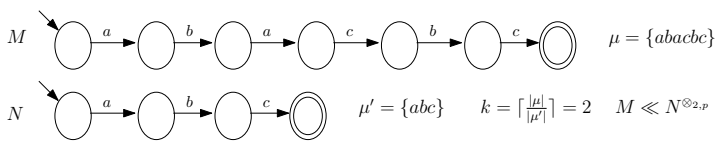


Fig. 5. An example of **SHUFFLE PRODUCT** problem.

Another factor of complexity of the WSC problem is the number of hybrid states in the available services. We investigate next the effect of this parameter on the complexity of the WSC problem.

5 Bounded number of hybrid states

The presence of hybrid states is a source of complexity in a WSC problem. As mentioned before, the size of intermediate states witnesses in configurations of $\mathcal{R}^{\otimes}$ used to simulate M is bounded by $norm(M)$. We are however unable to provide a similar bound for the number of hybrid states witnesses.

Figure 6 is an example of simulation between an FSM M and a PCSM $\mathcal{R}^{\otimes}$. The FSMs in $\mathcal{R}$ contain two hybrid states (state 1 and 2) and no intermediate state. Hence, a configuration of $\mathcal{R}^{\otimes}$ is a pair of integers witnessing the number of tokens in state 1 and state 2. The example illustrates the different roles that an hybrid state of $\mathcal{R}$ can play to simulate a state of M. Indeed an hybrid state of $\mathcal{R}$, can be used as:

(i) a terminal state, e.g., when testing whether $q_5 \ll (1,1)$, we can consider the second hybrid state of $\mathcal{R}$ as a terminal state and terminate the test, or

(ii)an intermediate state, e.g., when testing whether $q_2 \ll (1,1)$, the second hybrid state of $\mathcal{R}$ here plays the role of intermediate state, or

both a terminal and an intermediate state, e.g., when testing whether $q_1 \ll (1,0)$, a transition of $\Phi_{\mathcal{R}^{\otimes}}$ labeled by $(b,(-1,0))$ only appears in one branch in the simulation tree $\mathcal{T}_{sim}(M,\mathcal{R}^{\otimes})$. Hence, the first hybrid state of $\mathcal{R}^{\otimes}$ is considered intermediate in one branch and terminal in the other, or

a hybrid state, e.g., when it is used to simulate an hybrid state of $H(M)$.

We consider in the following the problem defined below.

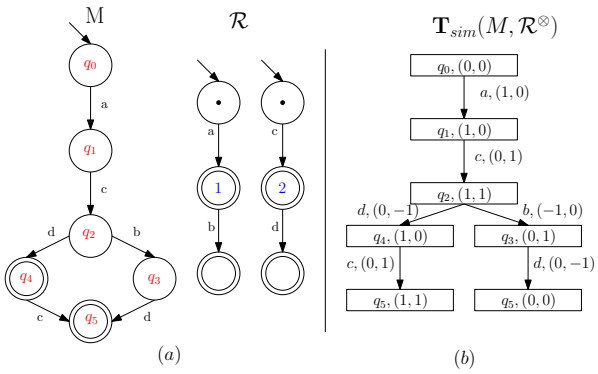


Fig. 6. Example of the simulation tree

Problem 4. **Unbounded Composition With limited number of Hybrid States** $UCHS(M, \mathcal{R}, k)$

Input : k an integer; $\mathcal{R}$ a set of FSMs, containing at most k hybrid states; M a target FSM.
Question : $M \ll \mathcal{R}^{\otimes}$?

It is worth noting that $UCHS(M, \mathcal{R}, k+1)$ is harder then $UCHS(M, \mathcal{R}, k)$. In the sequel, we progressively investigate the complexity of $UCHS(M, \mathcal{R}, k)$ problem for $k = 0$, then for $k = 1$ and finally for $k = 2$.

5.1 Case of composition without hybrid states (i.e. $k = 0$)

In this section, we are interested in the problem $UCHS(M, \mathcal{R}, 0)$. We first give a polynomial transformation, denoted $\mathcal{K}$, which is used to reduce $BC(M, \mathcal{R}, 1)$ to $UCHS(N, \mathcal{R}', 0)$. This transformation provides a mean to bound the number of instances used to prove simulation.

Definition 8. ***Transformation*** $\mathcal{K}$. *For an FSM* $M = (\Sigma_M, Q_M, F_M, q_0^M, \delta_M)$ *and a set of FSMs* $\mathcal{R} = \{M_1, ..., M_m\}$, *we define* $\mathcal{K}(M, \mathcal{R})$=$(N, \mathcal{R}'$=$\{N_1, .., N_m\})$ *where:*

1. *Each* N_i *is built based on* M_i, *by adding a letter* t_i *to its alphabet, a final state* f_i *and a transition set* $\{(q_0^{M_i}, t_i, f_i)\} \cup \{(q, t_i, f_i) | q \in F_{M_i}\}$. *All final states of* M_i *become intermediate in* N_i.
2. *N is defined as:*
   - $\Sigma_N = \Sigma_M \cup \{t_i | 1 \leq i \leq m\}$;
   - $Q_N = Q_M \cup \{r_i | 1 \leq i \leq m\}$;
   - $F_N = \{r_m\}$;
   - $\delta_N = \delta_M \cup \{(q, t_1, r_1) | q \in F_M\} \cup \{(r_i, t_{i+1}, r_{i+1}) | 1 \leq i < m\}$.

Figure 7 illustrates an example of this transformation. We prove later in proposition 2 that $\mathcal{K}$ defines a polynomial reduction of $BC(M, \mathcal{R}, 1)$ to $UCHS(N, \mathcal{R}', 0)$. In fact, the intuition behind this reduction is based on two points:

- By adding the sequence of letters $t_1, ..., t_m$ at the end of every execution accepted by N and adding t_i at the end of every execution accepted by $N_i \in \mathcal{R}'$, we ensure that even in an unbounded instances simulation, we can not use more than one instance of every N_i in order to simulate N.
- The construction of $\mathcal{R}'$ verifies that every hybrid state in $M_i \in \mathcal{R}$ becomes intermediate in N_i, while keeping its dual role: either terminate the execution by adding the letter t_i to the execution of $N_i$and reaching the terminal state f_i, or keep the execution in the same way as M_i.

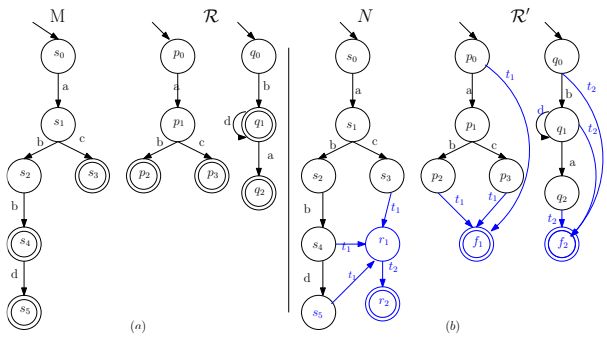


Fig. 7. *An example of transformation $\mathcal{K}$*

The following propositions show that the transformation $\mathcal{K}$ preserves the simulation preorder.

Proposition 1. *Let M be an FSM, $\mathcal{R} = \{M_1, ..., M_m\}$ be a set of FSMs and $\mathcal{K}(M, \mathcal{R}) = (N, \mathcal{R}' = \{N_1, .., N_m\})$. For p and q two states of respectively M and $\mathcal{R}^{\otimes_1}$, we have: $p \ll_{(M,(\mathcal{R})^{\otimes_1})} q$ iff $p \ll_{(N,(\mathcal{R}')^{\otimes_1})} q$.*

Proof. By construction of $\mathcal{K}(M, \mathcal{R})$, *if* $p \ll_{(M,\mathcal{R}^{\otimes_1})} q$ and p is terminal in M *then* $p \ll_{(N,(\mathcal{R}')^{\otimes_1})} q$.

We suppose next that:

If $(p, a, p') \in \delta_M$, $(q, a, q') \in \delta_{\mathcal{R}^{\otimes_1}}$ and $p' \ll_{(M,\mathcal{R}^{\otimes_1})} q'$, *then* $p' \ll_{(N,(\mathcal{R}')^{\otimes_1})} q'$.

and prove that $p \ll_{(N,(\mathcal{R}')^{\otimes_1})} q$.

For each $(p, a, p') \in \delta_N$, we have:

- *if* $a \in \Sigma_M$, *then* there exists $(q, a, q') \in \delta_{\mathcal{R}^{\otimes_1}} \subseteq \delta_{(\mathcal{R}')^{\otimes_1}}$ such that $p' \ll_{(N,(\mathcal{R}')^{\otimes_1})} q'$.
- else $a = t_1$, $p' = r_1$ and q is a product of final states of $\mathcal{R}$. therefore, there exists $(q, t_1, q') \in \delta_{(\mathcal{R}')^{\otimes_1}}$ such that $q' = (f_1, {q'}_{i_1}, ..., {q'}_{i_l})$ where ${q'}_{i_j}$ is final in $\mathcal{R}$ such that $p' \ll_{(N,(\mathcal{R}')^{\otimes_1})} q'$.

We conclude that if $p \ll_{(M,\mathcal{R}^{\otimes_1})} q$ then $p \ll_{(N,(\mathcal{R}')^{\otimes_1})} q$.

Reciprocally, we have $(p, a, p') \in \delta_N$ (respectively $\delta_{(\mathcal{R}')^{\otimes_1}}$) and $a \notin \{t_i | 1 \leq i \leq m\}$ iff $(p, a, p') \in \delta_M$ (respectively $\delta_{\mathcal{R}^{\otimes_1}}$). In addition, the definition of $\mathcal{K}$ ensures that if p is final in M and $p \ll_{(N,(\mathcal{R}')^{\otimes_1})} q$ then q is final in $\mathcal{R}^{\otimes_1}$. Hence if $p \ll_{(N,(\mathcal{R}')^{\otimes_1})} q$ then $p \ll_{(M,\mathcal{R}^{\otimes_1})} q$. □

In particular, we take p as the initial state of M and q the initial state of $\mathcal{R}^{\otimes_1}$. This implies that:

Proposition 2. *Let M be an FSM, $\mathcal{R} = \{M_1, ..., M_m\}$ be a set of FSMs and $\mathcal{K}(M, \mathcal{R}) = (N, \mathcal{R}' = \{N_1, .., N_m\})$. We have: $M \ll \mathcal{R}^{\otimes_1}$ iff $N \ll (\mathcal{R}')^{\otimes}$.*

Proof. We have $N \ll (\mathcal{R}')^{\otimes_1}$ *iff* $N \ll (\mathcal{R}')^{\otimes}$. Indeed, each path that starts from the initial state to a final one in N contains exactly one transition labelled by t_i, for each $i \in [1, m]$ and a similar path in each N_i contains exactly one transition labelled by t_i. □

Hence, $\mathcal{K}$ is a polynomial reduction of $BC(M, \mathcal{R}, 1)$ problem to the UCHS problem. This enables to derive the following result.

Theorem 4. *$UCHS(M, \mathcal{R}, 0)$ problem is Exptime-complete.*

Proof. According to proposition 2, the $\mathcal{K}$ transformation reduces $BC(M, \mathcal{R}, 1)$ to $UCHS(M, \mathcal{R}, 0)$ in polynomial time. Thus $UCHS(M, \mathcal{R}, 0)$ is Exptime-hard. Since it is also proven Exptime in [9], then $UCHS(M, \mathcal{R}, 0)$ is Exptime-complete. □

5.2 Case of composition with one hybrid state

We consider the problem $UCHS(M, \mathcal{R}, 1)$ where M is an FSM and $\mathcal{R}$ a set of FSMs containing at most one hybrid state ($n_H(\mathcal{R}) \leq 1$). We denote $k_0 = |Q_M|.2^{n_I(\mathcal{R}).log(norm(M))}$. Two nodes (q, c) and (q', c') in a simulation tree are called comparable if $q = q'$ and either $c \lhd c'$ or $c' \lhd c$. The nodes (q, c) and (q', c') are said incomparable otherwise.

Property 3. Let $\mathcal{R}$ be a set of FSMs containing at most one hybrid state. Two configurations of $\mathcal{R}^{\otimes}$ are comparable by the cover relation, *iff* they have exactly the same intermediate witnesses.

Proof. Acoording to property 2, for c, c' two configurations in $\mathcal{R}^{\otimes}$ we have $c \lhd c'$ *iff*:

1. c and c' have the same intermediate witnesses; and

2. for every hybrid witness $c[h]$, we have: $c[h] \leq c'[h]$.

In the current case, we consider that $\mathcal{R}$ has at most one hybrid witness. Hence, for any pair of configurations of $\mathcal{R}^{\otimes}$, condition 2 is verified.

We conclude that for every two configurations c, c' in $\mathcal{R}^{\otimes}$, $c \triangleleft c'$ *iff* c and c' have the same intermediate witnesses. □

Property 4. Let S be a set of nodes of $\mathcal{T}_{sim}(M, \mathcal{R}^{\otimes})$ that are pairwise incomparable, *then* $|S| \leq k_0$.

Proof. In configurations considered in $\mathcal{T}_{sim}(M, \mathcal{R}^{\otimes})$, intermediate witnesses are bounded by $norm(M)$ (property 1). Therefore and according to property 3, the number of incomparable configurations considered in $\mathcal{T}_{sim}(M, \mathcal{R}^{\otimes})$ is at most $2^{n_I(\mathcal{R}).log(norm(M))}$. Since $S \subset Q_M \times \mathcal{C}_{\mathcal{R}^{\otimes}}$, then $|S| \leq k_0$. □

Proposition 3. *If* $n_H(\mathcal{R}) \leq 1$, *then foreach* $(q, c) \in \mathcal{T}_{sim}(M, \mathcal{R}^{\otimes})$, $c[h = (n_I(\mathcal{R}) + 1)] \leq {k_0}^2$.

Proof. let P be a path in $\mathcal{T}_{sim}(M, \mathcal{R}^{\otimes})$, Int be an interval in $\mathbb{N}$ and $S = (v_n = (q_n, c_n))_{n \in Int}$ be a sequence of nodes in P such that:
-v_i is the ith node met in P that is comparable to one of its predecessors $v = (q_i, c)$; and
-For each $i, j \in Int$, v_i and v_j are incomparable.

If $Int = \emptyset$, then all nodes of P are not comparable. The size of P is then bounded by k_0, therefore, $c[n_I(\mathcal{R}) + 1] \leq k_0$ for each (q,c) in P.

We suppose next that $Int \neq \emptyset$ and take $Int = [1, k]$, $k \in \mathbb{N}$. We prove recursively that for each $l \in [1, k]$, $c_l[h] \leq l.k_0$.

For $l = 1$, we have $c_1[n_I(h] \leq k_0$.
For $1 < l < k$, we suppose that $c_l[h] \leq l.k_0$. Each node $v = (q, c)$ between v_l and v_{l+1} in P is either:

1. comparable to a node v_i with $i \in [1, l]$. In this case, $c[h] < c_i[h] \leq l.k_0$ (otherwise v should be a leaf).
2. incomparable to all its predecessors. The number of such nodes is bounded by k_0. And since transitions displacements is in $\{-1, 0, 1\}^h$, then we have $c[h] < l.k_0 + k_0$.

Therefore $c_{l+1}[h] \leq (l + 1).K_0$.

Once we reach v_k, each one of its possible successors $v = (q, c)$ is comparable to a node v_i with $c[h] < c_i[h]$, except for the last one that is the leaf of P.

Finally, since $k < k_0$ (because S is a sequence of incomparable nodes), we conclude that each node of P is in $Q_A \times ([1, norm(A)]^I \times [1, {k_0}^2])$. □

Since deciding simulation only requires to visit a node once, we argue next that this problem is in APspace (i.e a problem that can be solved by an alternating Turing machine in polynomial space): the size of a position of the simulation tree is polynomial in the input size (Proposition 3). Hence a polynomial space alternating turing machine can solve this simulation problem: universal states correspond to the target's and existential states correspond to the shuffle product's configurations. Note that APspace corresponds to Exponential time complexity. Given the above, we conclude that:

Lemma 2. $UCHS(M, \mathcal{R}, 1)$ *is in Exptime.*

To prove the Exptime-hardness of the problem, we recall that $UCHS(M, \mathcal{R}, 0)$ is Exptime-hard (theorem 4) and that $UCHS(M, \mathcal{R}, 1)$ is harder than $UCHS(M, \mathcal{R}, 0)$.

Theorem 5. $UCHS(M, \mathcal{R}, 1)$ *is Exptime-complete.*

5.3 Case of composition with two hybrid states

In this section, we consider the problem of unbounded composition of web services with at most 2 hybrid states in $\mathcal{R}$, *i.e.* $UCHS(M, \mathcal{R}, 2)$. Our approach is based on relating this simulation problem to the reachability issue.

For $x \in \mathbb{N}^{i_1}$ and $y \in \mathbb{N}^{i_2}$, we denote the concatenation of two vectors x and y, $(x.y) \in \mathbb{N}^{i_1+i_2}$ such that:

$$(x.y)[j] = \begin{cases} x[j] \; if \; j \in [1, i_1] \\ y[j] \; if \; j \in [i_1 + 1, i_1 + i_2] \end{cases}$$

We define the 2-dimension Vector Addition System with States (VASS) [11] $\mathcal{V}_{M,\mathcal{R}}$ as follows:

- States: $S \subseteq Q_M \times \{c \in \mathbb{N}^{n_I(\mathcal{R})} | \sum_{i=1}^{i=n_I(\mathcal{R})} c[i] \leq norm(M)\}$.
- Transitions: $W \subseteq S \times S \times \{-1, 0, 1\}^2$ such that:
  $((q, c), (p, d), x) \in W$ *iff* there exists $a \in \Sigma_M$, $y \in \mathbb{N}^2$ such that $(p, a, q) \in Q_M$ *and* $((c, y), a, (d, y + x)) \in \Phi_{\mathcal{R}\otimes}$ and $\sum_{i=1}^{i=n_I(\mathcal{R})} c[i] \leq norm(M)$ and $\sum_{i=1}^{i=n_I(\mathcal{R})} d[i] \leq norm(M)$.
- Initial configuration: the system starts with the state $(q_0^M, \{0\}^{n_I(\mathcal{R})})$ and the vector $(0, 0)$.

Figure 8 depicts an example of a VASS associated to an FSM M and a set of FSMs $\mathcal{R}$.

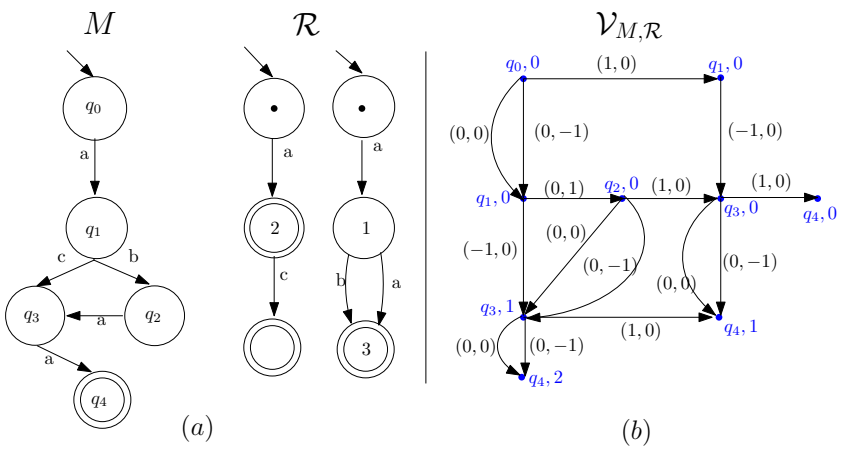


Fig. 8. *An example of a VASS associated to an FSM M and an FSMs set.*

The reachability issue in 2-dimension VASSs has been investigated by Hopcroft and Pansiot [11] in the general case where displacements are in $\mathbb{N}^2$. [11] gives an algorithm to prove the semi-linearity of the reachability set of such systems. The algorithm builds a tree T_{reach} labeled by 3-tuples (p, c, A_c) where p is the current state, $c \in \mathbb{N}^2$ is a vector reached in the system and $A_c \subset \mathbb{N}^2$. (p, c, A_c) denotes that every vector in the linear set $\{c + \alpha_1 a_1 + ... + \alpha_n a_n | i \in [1, n], a_i \in A_c \text{ and } \alpha_i \in \mathbb{N}\}$ can be reached in state p from the initial configuration.

We consider in the following a simulation tree $T_{sim}(M, \mathcal{R}^{\otimes})$=$(V, v_0, E)$, a recheability tree $T_{reach}(\mathcal{V}_{M,\mathcal{R}})$=$(V', v'_0, E')$ and a function π defined as follows:

$$\begin{array}{rccc} \pi : & V' & \rightarrow & V \\ & ((p, c), x, A_x) & \mapsto & (p, (c.x)) \end{array}$$

The following proposition enables to establish a connection between paths in a simulation tree and a corresponding recheability tree.

Proposition 4. *Let $\mu = v_0...v_t$ be a path in $T_{sim}(M, \mathcal{R}^{\otimes})$. Then there exists a path $\mu' = v'_0...v'_t$ in $T_{reach}(\mathcal{V}_{M,\mathcal{R}})$ such that $v_i = \pi(v'_i)$, $i \in [0, t]$.*

Proof. We proof by induction on the length i of the path $\mu = v_0...v_t$.

For $i = 0$ we have $v_0 = (q_0^M, c_0) = \pi((q_0^M, \{0\}^{n_I(\mathcal{R})}), x_0, A_{c_0}) = \pi(v'_0)$.

Now suppose that the property is true for $i < t$ and $v_0...v_{i+1}$ is a path in $T_{sim}(M, \mathcal{R}^{\otimes})$. Then by hypothesis there exists a path $v'_0...v'_i$ in $T_{reach}(\mathcal{V}_{M,\mathcal{R}})$, such that $v_j = \pi(v'_j)$, $j \in [0, i]$.

Suppose that v'_i is a leaf in $T_{reach}(\mathcal{V}_{M,\mathcal{R}})$. Then according to the algorithm of Hopcroft and Pansiot [11], we have either:

- There exist $j \in [0, i-1]$ such that $v'_j = ((p,c), y, A_x)$ and $v'_i = ((p,c), x, A_x)$ with $y \leq x$ (see Algorithm **??**, line 1). This implies that $v_j \triangleleft v_i$, which contradicts that v_i is not a leaf in $T_{sim}(M, \mathcal{R}^{\otimes})$, i.e. $(c,y) \triangleleft (c,x)$.
- There is no transition from v'_i in the system (see Algorithm **??**, line 1). But for $v_i = (p,(c,x))$ and $v_{i+1} = (q,(d,y))$ we have $v_i v_{i+1} \in E$ which means that $(p,a,q) \in \delta_M$ and $((c,x),a,(d,y)) \in \Phi_{\mathcal{R}^{\otimes}}$. This implies that $((p,c),(q,d),y-x) \in W$. Contradiction.

Therefore, we have : $v'_{i+1} = ((q,d), y, A_y)$ is a successor of v'_i in $T_{reach}(\mathcal{V}_{M,\mathcal{R}})$, with $v_{i+1} = (q,(d,y))$. We conclude that μ' is a path in $T_{reach}(\mathcal{V}_{M,\mathcal{R}})$. □

The following corollary is a consequence of Proposition 4.

Corollary 2. *$T_{sim}(M, \mathcal{R}^{\otimes})$ is a sub-tree of $T_{reach}(\mathcal{V}_{M,\mathcal{R}})$.*

Clearly the time complexity for computing $T_{sim}(M, \mathcal{R}^{\otimes})$ is dominated by the complexity of computing $T_{reach}(\mathcal{V}_{M,\mathcal{R}})$. Moreover we know from [12] that the size of $T_{reach}(\mathcal{V}_{M,\mathcal{R}})$ is in 2-Exptime. Hence, we derive the following complexity result.

Theorem 6. *$UCHS(M, \mathcal{R}, 2)$ is in 3-Exptime.*

Proof. According to [12], the size of $T_{reach}(\mathcal{V}_{M,\mathcal{R}})$ is of order $O(2^{2^{\alpha}})$ where $\alpha = max(|S|, |W|) \leq c \times (|Q_M| \times Norm(M)^{n_I(\mathcal{R})})^2$ with c is a constant. Then according to Corollary 2, the size of $T_{sim}(M, \mathcal{R}^{\otimes})$ is bounded by $2^{2^{2^{c_1 + c_2 * \beta}}}$ where c_1 and c_2 are constants and $\beta = log(|Q_M|) + n_I(\mathcal{R}) \times log(Norm(M))$. □

Our proof for Theorem 6 can be seen more as an embedding of the search space explored by a simulation test to the one explored when the reachability issue is considered. This is an approach that can not so far be generalized because the best upper bound provided for vector addition systems reachability is non-primitive recursive; in fact even the existence of a primitive upper-bound is still open [14].

6 Conclusion

In this paper we have considered two parameters that are source of complexity of the web services composition problem. We have shown that among the considered problems, several instances remain Exptime-complete when a parameter is bounded. It remains an open question to identify the complexity of $UCHS(M, \mathcal{R}, k)$ for any $k \in \mathbb{N}$; [5] proves in the context of Z-Reachability that the problem is k-Exptime. This complexity is quite far from the known lower bound (2-Exptime). It is also interesting to improve the polynomial complexity given for k=2 in [7] (polynomial of the 17th degree) and/or give a simpler algorithm that can eventually be extended to the general case.

References

1. Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables.* Dover, New York, ninth dover printing, tenth gpo printing edition, 1964.
2. Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services: Concepts, Architectures and Applications.* Springer, 2010.
3. B. Benatallah, F. Casati, and F. Toumani. Web Service Conversation Modeling: A Cornerstone for E-Business Automation. *IEEE Internet Computing*, 08(1):46–54, 2004.
4. D. Berardi, D.Calvanese, G. De Giacomo, R. Hull, and M. Mecella. Automatic composition of transition-based semantic web services with messaging. In *VLDB*, pages 613–624, 2005.
5. Tomáš Brázdil, Petr Jančar, and Antonín Kučera. Reachability games on extended vector addition systems with states. In *Automata, Languages and Programming*, pages 478–489. Springer, 2010.
6. T. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification: a new approach to design and analysis of e-service composition. In *WWW'03*. ACM, 2003.
7. Jakub Chaloupka. Z-reachability problem for games on 2-dimensional vector addition systems with states is in p. In *Reachability Problems*, pages 104–119. Springer, 2010.
8. Jean-Baptiste Courtois and Sylvain Schmitz. Alternating vector addition systems with states. In *Mathematical Foundations of Computer Science 2014*, pages 220–231. Springer, 2014.
9. Ramy Ragab Hassen, Lhouari Nourine, and Farouk Toumani. Protocol-based web service composition. In *ICSOC*, pages 38–53, 2008.
10. Monika Rauch Henzinger, Thomas A Henzinger, and Peter W Kopke. Computing simulations on finite and infinite graphs. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 453–462. IEEE, 1995.
11. John Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *TCS*, 8(2):135–159, 1979.
12. Rodney R Howell, Louis E Rosier, Dung T Huynh, and Hsu-Chun Yen. Some complexity bounds for problems concerning finite and 2-dimensional vector addition systems with states. *TCS*, 46:107–140, 1986.
13. Joana Jedrzejowicz and Andrzej Szepietowski. Shuffle languages are in p. *Theoretical Computer Science*, 250(1-2):31–53, 2001.
14. S Rao Kosaraju. Decidability of reachability in vector addition systems. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 267–281. ACM, 1982.
15. Slawomir Lasota. Expspace lower bounds for the simulation preorder between a communication-free petri net and a finite-state system. *Inf. Process. Lett.*, 109(15):850–855, 2009.
16. Anca Muscholl and Igor Walukiewicz. A lower bound on web services composition. *Logical Methods in Computer Science*, 4(2), 2008.

GPU computations and memory access model based on Petri nets*

Anna Gogolińska, Łukasz Mikulski, and Marcin Piątkowski

Faculty of Mathematics and Computer Science,
Nicolaus Copernicus University, Toruń, Poland
{anna.gogolinska,lukasz.mikulski,marcin.piatkowski}@mat.umk.pl

Abstract. In modern systems CPUs as well as GPUs are equipped with multi-level memory architectures, where different levels of the hierarchy vary in latency and capacity. Therefore, various memory access models were studied. Such a model can be seen as an interface abstracting the user from the physical architecture details. In this paper we present a general and uniform GPU computation and memory access model based on bounded inhibitor Petri nets (PNs). Its effectiveness is demonstrated by comparing its throughputs to practical computational experiments performed with the usage of Nvidia GPU with CUDA architecture.
Our PN model is consistent with the workflow of multithreaded GPU streaming multiprocessors. It models a selection and execution of instructions for each warp. The three types of instructions included in the model are: the arithmetic operation, the access to the shared memory and the access to the global memory. For a given algorithm the model allows to check how efficient the parallelization is, and whether a different organization of threads will improve performance.
The accuracy of our model was tested with different kernels. As the preliminary experiments we used the matrix multiplication program and stability example created by Nvidia, and as the main experiment a binary version of the least significant digit radix sort algorithm. We created three implementations of the algorithm using CUDA architecture, differing in the usage of shared and global memory as well as organization of calculations. For each implementation the PN model was used and the results of experiments are presented in the work.

Keywords: Petri nets, CUDA architecture, GPU, memory model

1 Introduction

The inter-process communication over a common part of the memory shared by processes is a usual performance bottleneck in multiprocessor environments. In modern systems CPUs as well as GPUs are equipped with multi-level memory architectures, where different levels of the hierarchy vary in latency and capacity.

* This research has been supported by the Polish National Science Center through grant No.2013/09/D/ST6/03928

By considering different local views of the processes on the common part of the memory one can try to improve the processor utilization. Therefore, various memory access models were studied, see for instance [8, 11, 15]. Such a model can be seen as an interface abstracting the user from the physical architecture details. It allows to specify, without a reference to processors, the local views that are possible in concurrent task executions and maintain its consistency.

Another important issue is the task distribution between threads and CPU/GPU cores and the instruction scheduling, which can have a significant impact on the efficiency. Consider for instance running the three threads on a single processor depicted in Figure 1. Each of them performs a list of arithmetic operations interleaved by memory reads/writes consisting of a short preprocessing and then a longer period of waiting for the memory access (which is a usual situation in parallel computing). In the initial part of the computation each thread realizes its preprocessing for the memory access and then starts to wait for the access itself. This causes the processor idle period when all threads are waiting (marked as I). After that the arithmetic operations of all threads are executed simultaneously. They can be scheduled in such a way that one thread waits for the memory access while other threads perform their computations and there is no idle period (marked as II). Thanks to that the waiting period of a thread can be hidden behind the active computations of other threads.

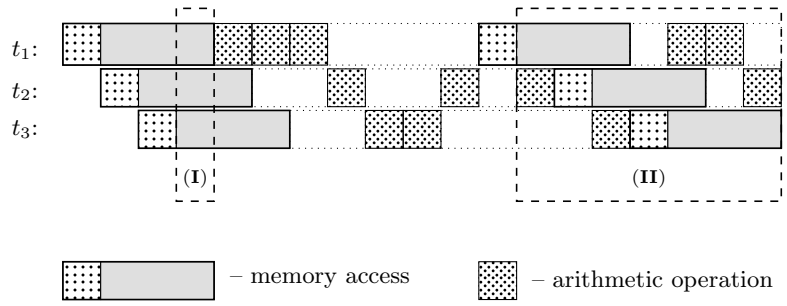


Fig. 1. The example run of the threads t_1, t_2 and t_3 on a single processor. The area marked with (I) represents the idle period when no computation is done, and the area marked with (II) represents the period when the waiting for the memory access is hidden behind arithmetic operations.

The main contribution of this paper is a general and uniform GPU computation and memory access model based on bounded Petri nets [14] together with the application simulating its execution. For a given algorithm pseudocode (or the source code) one can use our model to count the number of operations performed (taking into account their simultaneous execution). The main advantage of the model over the basic arithmetic counting and other models described below is the possibility of reorganization of parts of the code, the potential ability to predict a duration of GPU calculations and to handle the aforementioned

computation scheduling. Such an approach might be used to improve the algorithm code organization and to partition the computation tasks between the threads to maximally increase the efficiency.

The effectiveness of our model is demonstrated by comparing its throughputs to practical computational experiments performed with the usage of Nvidia GPU. We study the impact on the complexity of various parameters such as the number of concurrent processes, and the level of memory used (shared memory vs. global memory). The successful application of our model is discussed in details, as a proof of concept, on a single example of digit radix sorting algorithm. We do not want to discuss methods of parallelization, nor the most efficient way of using different types of the memory. It is beyond the scope of this paper. Those problems are very complex, and cannot be explained in such a short publication. More about those topics can be found for instance in [6, 20].

Our PN model is not the only one GPU efficiency model. There are quite a few other GPU performance models, which can be divided into two groups (according to [12]):

(1) Calibrated performance models that make specific predictions and include many lower levels of details. They usually contain many specialized parameters, some of which may be difficult to obtain or calculate. The first example is a model presented in [7]. It is the first analytical model of the GPU efficiency. The most important values used there are MWP (number of memory requestes that can be executed concurrently) and CWP (number of warps, which can be computed while one warp is waiting for memory values). The model consists of 14 equations, which contains 21 parameters. Other example is a model presented in [19]. The authors created not only performance GPU model but also power consumption model. In the performance part they estimate execution time for individual GPU architecture components (e.g. shared, global memory) separately. This way they easily identify potential performance bottlenecks. The model has a form of equations and contains 32 parameters.

(2) Asymptotic models for algorithm analysis at a high level of abstraction that capture only the essential features of GPU architecture. One of the models from this group is the model described in [13]. He used elements of PRAM, BSP and QRQW approaches. The model calculates the number of cycles required for the whole kernel, taking into account the number of blocks, warps and threads, the maximal number of cycles required by a single thread to perform calculations and the number of threads which can be executed in parallel. However the model is quite simple and some important elements are neglected (like hiding memory latency in computations). The other asymptotic model is Thread Multi-Core Memory (TMM) model presented in [11]. Basing on the TMM, GPUs can be presented as abstract core groups, each containing a number of cores and fast local memory. A large and slow global memory is shared by all cores. The running time of the algorithm is calculated basing on suitable equation with basic GPU parameters. One of the most popular GPU efficiency models is the roofline model introduced in [21]. It can be used to obtain performance estimates of GPU computations, and requires two parameters: the number of operations

performed by a kernel and the number of bytes transferred from/to the memory, which can be used to calculate the arithmetic intensity I. In the naive roofline model I can be handily presented as a point in two-dimensional space restricted by two ceilings lines: the memory bandwidth and the processor's peak efficiency. The resulting performance is a bound under which the arithmetic intensity appeared: the memory bandwidth bound or the peak performance bound. In the extended versions of the model, additional ceilings can be added. They related for example to software prefetching or task level parallelism. The roofline model can determine the type of kernel limitation and show, how optimal the program is. However, such a simple arithmetic operation cannot precisely represent the complex processes of kernel execution, like for example hiding the waiting period in calculations (see Figure 1). Similar problem occurs in other purely arithmetical models. More complex tools are necessary for such a purpose. Our model belongs to the asymptotic group, and to the best of our knowledge it is the first one utilizing Petri nets.

The paper is organized as follows. In the next section we describe Graphical Processing Units and CUDA Toolkit, focusing in particular on memory types (and organization). In Section 3 we recall some standard notions and notations related to Petri nets. Then we introduce our memory access and computation model followed by the description of radix sort algorithm. We also present the results of experiments conducted on the base of our implementations of the radix sort algorithm. We conclude the paper and give some directions of further research in the last section.

2 CUDA

An intensive development of Graphical Processing Units (GPU in short) resulted in construction of high performance computational devices, which besides the graphical display management, allow also the execution of parallel general purpose algorithms (not necessarily related to computer graphics). As opposed to CPU consisting of a few cores optimized for sequential serial processing, GPU contains thousands of smaller, more efficient cores designed for handling multiple tasks simultaneously. The most popular ones are GPU's produced by Nvidia Corporation, supplied with Nvidia CUDA Toolkit (see [2]).

A program running in heterogeneous environment equipped with GPU can be split into so-called *host* parts, which are executed by CPU, and so-called *kernel* parts, which are executed by GPU. The host part specify the kernel execution context and manages the data transfer between the host and the GPU memory. The kernel functions create a big number of threads allowing a highly parallel computation (where each thread runs the same kernel code). GPU threads, as opposed to CPU threads, are much lighter, hence their creation and controlling requires less CPU cycles.

All threads executed on GPU are organized into several equally-sized thread blocks, which in turn are organized into a grid.

A thread block is the set of concurrently executed threads. Such an execution and the thread cooperation can be coordinated by a barrier synchronization. Moreover, the data can be exchanged between threads using a shared memory. The size of a block is limited by the capacity of resources accessible on a single processor core. On currently available GPU's a single block can contain up to 1024 threads. Each thread block within a grid is uniquely identified by its block ID, and each thread within a block – by its thread ID.

A grid is an array of thread blocks executing the same kernel. The number of blocks in a grid is specified by the amount of data to be processed and the number of available processors. The exchange of data between threads within a grid requires the usage of the global memory. Moreover, while all threads within a single block run simultaneously, different blocks can be executed in any order.

The physical Nvidia GPU architecture consists of several multithreaded streaming multiprocessors (SM in short). Thread blocks are distributed between SM's in such a way that all threads within a single block are concurrently executed on the same SM (different blocks may, but not necessarily have to, be executed on the same SM). A streaming multiprocessor organizes threads into so-called *warps* consisting of 32 threads each. The partition is done according to increasing thread ID. Each SM works utilizing SIMT (single-instruction, multiple-threads) architecture, which means that all threads within a single warp execute one common instruction at a time. Any divergence (e.g. caused by a data processed) leads to a serial execution of a single computation path until possible convergence to the same execution path.

A single SM serves multiple warps. It is equipped with a number of warp schedulers and instruction dispatch units (currently 2 or 4 depending on the device used). A scheduler selects a warp, which is ready to be executed and issues it to the physical cores of GPU. If the currently active warp needs to wait for the memory read/write operation it is replaced by another ready warp. While the replaced warp is waiting for the memory access, other warps perform their computations, therefore the SM is busy as often as possible. The waiting period of a single warp is hidden behind the computations of the others (if there are only enough active warps available) and is not seen outside the SM. The simulation of such a behavior is the main part of our model.

The above mentioned heterogeneous environment is equipped with the *host memory* managed by CPU and the GPU *device memory*. The latter is significantly more complex than the former. Due to a necessary compromise between the data transfer/access speed and the possible capacity, the GPU device memory consists of various types of data storage, such as global, constant, local and shared memory.

The global memory is the largest and at the same time the slowest type of GPU memory (with hundreds of cycles latency). Together with the constant memory it is the only type of GPU memory, which can be accessed by the host. It is available for reading and writing for all running threads, however the data exchange and result sharing are possible only after a kernel-wide global synchronization.

The content of the global memory is accessed in blocks of size 32, 64 or 128 bytes (depending on the device used). Every time an element within a block is accessed, the whole block have to be transferred. Therefore, the concurrent (among the threads within a single warp) global memory read and write operations are grouped into transactions, the number of which depends on the cache lines required to serve all threads within a warp. However, if different threads in a warp refer to different memory blocks, all such blocks have to be transferred to cache sequentially. Such a situation cause the necessity of repeated global memory accesses.

The shared memory is a fast memory physically placed inside a multiprocessor. It consists of blocks, each of which is available for all threads within a single thread block. Moreover, each such block is divided into several so-called memory banks. The access of different threads to different banks is realized simultaneously, while the access of different threads to the same bank is realized sequentially. Such a situation is called the *bank conflict*. The shared memory can be used for data exchange between threads within the same thread block after block-wide thread synchronization.

The local memory of a single thread consists of a number of registers, which are the fastest type of memory available (with almost negligible access time). It is used to store local thread variables. Due to large number of threads the capacity of each thread local memory is strongly limited.

To complete the picture we have to mention also the constant and texture memory – dedicated parts of the GPU device memory (usually buffered). Both of them are optimized for access speed within the device, but are available for threads in read-only mode. Neither of them is considered in our model.

3 Petri Nets

The set of non-negative integers is denoted by $\mathbb{N}$. Given a set X, the cardinality (number of elements) of X is denoted by $|X|$, the powerset (set of all subsets) by 2^X – the cardinality of the powerset is $2^{|X|}$. Multisets over X are members of $\mathbb{N}^X$, i.e., functions from X into $\mathbb{N}$. For convenience and readability, if the set X is finite, multisets in $\mathbb{N}^X$ will be represented by vectors of $\mathbb{N}^{|X|}$ (assuming a fixed ordering of the set X). The addition and the partial order $\leq$ on $\mathbb{N}^X$ are understood componentwise, while $<$ means $\leq$ and $\neq$.

Let us now recall basic definitions and facts concerning inhibitor Petri nets [1, 16].

Definition 1. *An inhibitor*[1] *place/transition net (p/t-net) is a quintuple* $S = (P, T, W, I, M_0)$*, where:*

- *P and T are finite disjoint sets, of places and transitions (actions), respectively;*

[1] Note that in the case of bounded nets the use of inhibitors is not necessary, one can provide an equivalent (with more complex structure) net without inhibitors.

- $W : P \times T \cup T \times P \to \mathbb{N}$ *is an arc weight function;*
- $I \subseteq P \times T$ *is an inhibition relation;*
- $M_0 \in \mathbb{N}^P$ *is a multiset of places, named the initial marking.*

For all $a \in T$ we use the following denotations:

$^\bullet a = \{p \in P \mid W(p, a) > 0\}$ – the set of *entries* to a
$a^\bullet = \{p \in P \mid W(a, p) > 0\}$ – the set of *exits* from a
$^\circ a = \{p \in P \mid (p, a) \in I\}$ – the set of *inhibitor places* for a.

Petri nets admit a natural graphical representation. Nodes represent places and transitions, arcs with classical arrow heads represent the weight function, while arcs with small circles as arrowheads represent inhibition relation. Places are indicated by circles, and transitions by boxes. Markings are depicted by tokens inside the circles, the capacity of places is unlimited. However, Petri nets used in our model are bounded (which means that there exist a common bound for all the numbers of tokens appearing during the computation in a single place).

The set of all finite strings of transitions is denoted by T^*, the empty string is denoted by ε, the length of $w \in T^*$ is denoted by $|w|$, number of occurrences of a transition a in a string w is denoted by $|w|_a$.

Multisets of places are called markings. In the context of p/t-nets, they are typically represented by nonnegative integer vectors of dimension $|P|$, assuming that P is totally ordered.

A transition $a \in T$ is enabled at a marking M whenever $^\bullet a \leq M$ (all its entries are marked) and $\forall_{p \in {}^\circ a}\ M(p) = 0$ (all inhibitor places are empty). If a is enabled at M, then it can be executed. A marking M is called a *dead marking* if no transition is enabled at M (which means that $\forall_{a \in T} \exists_{p \in P} \big(W(p, a) > M(p) \vee ((p, a) \in I \wedge M(p) > 0)\big)$). The execution of an enabled transition a is not forced and changes the current marking M to the new marking $M' = (M - {}^\bullet a) + a^\bullet$ (tokens are removed from entries, then put to exits). We shall denote Ma for the predicate "a is enabled at M" and MaM' for the predicate "a is enabled at M and M' is the resulting marking".

In this paper however, we use the maximal concurrent semantics and in every marking we execute one of the maximal sets of enabled transitions (i.e. a step, which is maximally concurrent at this marking). Formally, a set of transitions $A \subseteq T$ is called *step* and is enabled if $\big(\sum_{a \in A} {}^\bullet a\big) \leq M$ and $\forall_{p \in (\bigcup_{a \in A} {}^\circ a)}\ M(p) = 0$. The execution of a step A changes the current marking M to the new marking $M' = \big(M - \sum_{a \in A} {}^\bullet a\big) + \sum_{a \in A} a^\bullet$. We say that a step is *maximally concurrent at marking* M if A is enabled at M and $\forall_{a \notin A} A \cup \{a\}$ is not enabled at M.

The notions of enabledness and execution we extend, in a natural way, to strings of steps (*computations*): the empty string ε is enabled at any marking, a string $w = Av$ is enabled at a marking M whenever MAM' and v is enabled at M'. The predicates MA, Mw, MAM' and MwM' are defined like for single transitions.

4 Memory access and computations model

The Petri net model of GPU calculations is consistent with the workflow of multithreaded streaming multiprocessors (SMs). The model represents the way one SM operates. It models each warp assigned to the SM, selection of the next instruction for the SM, accesses to the global and shared memory, and arithmetic operations. It does not represent threads hierarchy (blocks, grid), repeated accesses to the global memory nor bank conflicts. The model allows simultaneous accesses to the global memory, but the number of warps, which can use the global memory, at the same time, is limited by the number of SM warp schedulers (2 or 4). Each element of the model was created basing on [3, 4].

The size of the considered Petri net depends on the number of warps. The two elements: the place p_0 – *SM* and the transition t_0 – *waiting* are the constant part of the model, other are generated for every warp. The place p_0 represents the streaming multiprocessor and its initial marking should correspond to the number of warp schedulers. For the modern graphical cards it should be 2 or 4. This place is connected by a loop with the transition t_0. The transition t_0 can be executed only when the warp schedulers cannot schedule any instruction, i.e. there is no instruction ready to be executed. The *waiting* transition is added to the model to gain control over how many steps of the calculations on the SM is idle. The minimization of that number is crucial for optimization of GPU programming. Beside those two elements, the PN model contains 18 places and 17 transitions for every warp required by the analysed algorithm. That part is called a warp part of PN model (WPNM). The detailed description of the most important places and transitions of WPNM is presented in Table 1.

The WPNM together with *SM* place and *waiting* transition are depicted in Figure 2. The place p_1 represents the activation of the warp. It is marked when the warp is active. The place p_2 is marked when the warp finishes the execution of its previous instruction. The warp can be scheduled for the execution only when the places p_2 and p_4 are marked. A token in p_4 means that the next instruction is selected and ready. The places from p_3 to p_{10} and the transitions t_2, t_3, t_4 (the frame I part in Figure 2) are responsible for controlling and selecting the instructions. The three types of instructions are allowed in the model: an arithmetic calculation, an access to the shared memory and an access to the global memory. The initial marking of the place p_5 corresponds to the number x_a of arithmetic operations required by the analyzed algorithm. Similarly, the initial markings of the places p_7 and p_9 represent respectively the numbers x_s and x_g of read/write operations from/to the shared and global memory. If the place p_3 is marked and at least one of the places: p_5, p_7, p_9 is not empty, the next instruction can be selected. The selection is random. When the instruction is chosen, according to its type (arithmetic calculation, shared memory access, global memory access) the marking of the corresponding place is decreased and a token is added to p_4. Moreover, (according to the instruction type) one of the places: p_6, p_8 or p_{10} is marked. Now the selected instruction is ready to be executed and the warp can be processed by SM. The execution of the instruction

is represented by the three parts of the net marked in frame II, frame III and frame IV (see Figure 2). They correspond to the type of the selected instruction: frame II for an arithmetic calculation, frame III – an access to the shared memory and frame IV – an access to the global memory. The arithmetic calculation is simply represented by one place and two transitions. When the calculation is done, tokens are put in places: p_2 and p_3 (which means that the instruction is finished and the next one can be selected), and in the place p_0 (which means that SM is ready to execute the next instruction). The same situation is obtained when the access to the memory (shared or global) is finished, but those parts of the model contain more places and transitions. Those additional elements are used to model the memory access latencies. The shared memory latency and the global memory latency are the parameters of the model and are denoted by l_1 for the global memory and by l_2 for the shared memory. The transition t_9 (t_{12} respectively) may be executed only after l_1 (l_2 respectively) executions of t_8 (t_{17} respectively). For testing, their default values were 20 for l_1 and 2 for l_2, which is consistent with [4].

The transition t_{16} is connected by inhibitor arcs with the places p_5, p_7 and p_9 (the frame I in Figure 2) and can be executed only when those places are empty, i.e. there is no instruction left for execution. The transition is also connected by a regular arc with p_2, Moreover, t_{16} is the only one able to take the token from the place p_1 and its execution is equivalent to the termination of a given warp.

As it was mentioned above, the WPNM (places from p_1 to p_{18} and transitions from t_1 to t_{17}) is generated for a single warp. In the case of multiple warps a separate WPNM should be generated for each of them. To distinguish between distinct WPNMs one can either increase the numeration of places and transitions accordingly or assign to them two-part labels consisting of the original place/transition number together with the warp id. It should be noticed that each place corresponding to p_2 and representing the readiness of the given warp should be connected by an inhibitor arc with the transition t_0. Moreover, each transition corresponding to t_1 should be connected with the place p_0 (SM). The same goes for transitions corresponding to t_5, t_{14} and t_{15}. Note that the control is returned by t_5 not t_{11} in the case of part responsible for the access to the global memory. Thanks to that, SM can process the next ready warp while the current one is waiting for the memory access.

Our PN model of GPU computation and memory access may by adapted for any algorithm. Instantiations of the model for different kernels may differ in the number of warps and the marking of places responsible for instructions counting (i.e. p_5, p_7 and p_9). For the chosen number of required warps, a new model containing sufficient number of WPNMs can be generated. The other possibility is the generation of one big model with the maximal possible number of WPNMs (i.e. 64 for modern graphical cards [4]), and then the necessary number of places representing active warps should be marked. The number of WPNMs can be calculated basing on the number of threads and blocks, which are parameters of the kernel. Notice that the model provides no controlling mechanism for the maximum number of WPNMs. It is up to the user to be aware

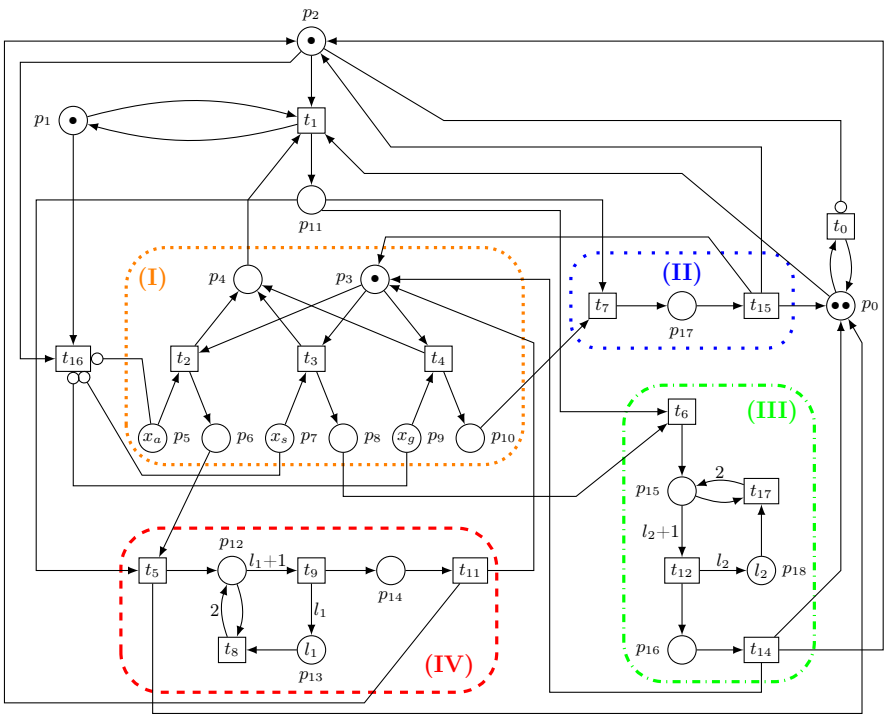


Fig. 2. The example of a warp PN model (WPNM). In frame I part the next instruction is selected. Sections: frame II, frame III and frame IV represent the execution of different types of instructions: arithmetic calculation, access to the shared memory and access to the global memory (respectively).

Place/Transition name and description			
p_0	Streaming Multiprocessor (SM)	t_0	Waiting
p_1	Active warp	t_1	The warp is executed on SM
p_2	The previous instruction is finished and the warp is ready for the next one	t_2	Selection of a calculation for the next instruction
p_3	Check the next instruction	t_3	Selection of an access to the shared memory for the next instruction
p_4	Next instruction is ready	t_4	Selection of an access to the global memory for the next instruction
p_5	Arithmetic operations	t_5	Execution of the access to the global memory
p_6	Instruction – arithmetic calculation	t_6	Execution of the access to the shared memory
p_7	Access to the shared memory	t_7	Execution of arithmetic operation
p_8	Instruction – shared memory access	t_8	Waiting for the global memory access
p_9	Access to the global memory	t_9	Access to the global memory
p_{10}	Instruction – global memory access	t_{11}	Access to the global memory is finished
		t_{12}	Access to the shared memory
		t_{14}	Access to the shared memory is finished
		t_{15}	Calculation is finished
		t_{16}	Termination of the warp
		t_{17}	Waiting for the shared memory access

Table 1. The description of the places and transitions depicted in the Figure 2.

that the maximal number of WPNMs is limited to 64 in modern GPUs. The number of arithmetic operations and accesses to the shared and global memory

need to be calculated for the considered algorithm. Those numbers should be used as the initial marking of places p_5, p_7 and p_9. If the model is constructed according to the description above, it is ready to be used out of the box. The usage of the model involves the execution of computations according to the maximal concurrent semantics (i.e. concurrent execution of all transitions that are enabled) starting from the initial marking until reaching the dead marking. The latter is obtained only when all places corresponding to p_1 (for each WPNM) become empty, which is equivalent to the termination of all warps. The number of steps of the computation is returned by the model and corresponds to the GPU execution time.

The maximal concurrent semantics requires execution of all enabled transitions. However, some of the enabled transitions may be in a conflict (i.e. execution of one transition makes disables another transition). In our implementation of the model, in every step, a permutation of the enabled transitions is randomly generated (using uniform distribution). Transitions are executed according to an order determined by the generated permutation. If two (or more) transitions are in a conflict, the transition appearing earlier in the considered order is executed.

The initial tests of the PN model were performed using the matrix multiplication kernel from [4] and the *stability* example from [20]. The PN models were generated for both kernels. The matrix multiplication program was executed many times with different sizes of matrices. Similarly, the *stability* example was executed with different values of *Time Step* and *Final Time* parameters. For the same data, the computations of the PN model were executed. The execution times of kernels and the numbers of steps of PN computations were compared. The results were consistent in both cases.

5 Experimental results

In the main experiment we used a binary version of the least significant digit radix sort algorithm [17, 18]. The idea of this method is to sort a list of positive n-bit integers using their binary representation. We make n runs rearranging the list in such a manner that in i-th run all the integers having 0 on i-th bit are arranged in the first part of the array, while those having 1 – in the second part. An important requirement is to preserve the order of elements which do not differ on the processed bit. In other words, the sorting subroutine need to be stable. As a side effect the whole sorting procedure is also stable.

In the parallel version we made n runs (one for every bit), each run consisting of three phases. At the beginning of each run, we partition the dataset equally between m nodes. During the first phase, j-th node counts $zeros[i, j]$ – the number of elements containing 0 on i-th bit (consequently, we know $ones[i, j]$ – the number of elements with 1 on i-th bit for this node). In the next phase, we need to compute the positions for the set of elements assigned to each node. Namely, j-th node should place all the elements having 0 on i-th bit between $\sum_{k<j} zeros[i, k]$ and $\left(\sum_{k\leq j} zeros[i, k]\right) - 1$, while those with 1 on i-th in the

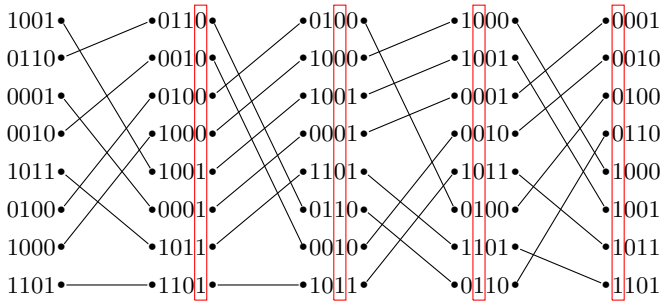


Fig. 3. Example of the use of radix sort procedure for four-bit integers. In consecutive columns we present the lists after each run of sorting subroutine. The rectangles emphasize columns with freshly sorted bits.

range

$$\Big[\sum_{k\le m} zeros[i,k] + \sum_{k<j} ones[i,k], \sum_{k\le m} zeros[i,k] + \big(\sum_{k\le j} ones[i,k]\big) - 1\Big].$$

The last, third phase, is the rearrangement of the list of integers. Each node traverse assigned part of the data splitting it into two parts (containing only 0 on i-th bit and only 1 on i-th bit) with the use of the positions computed in the second phase and in a stable manner. Since the output space for the nodes is partitioned into disjoint blocks, this phase may be realized using either shared or global memory.

We consider three CUDA implementations of the algorithm described above. In all versions, the array of integers A to be sorted is stored in the global memory. During each kernel execution, one block of threads (with different number of threads) is created. Each thread has its own part of the array A assigned. Its size is the parameter and is denoted by $memsize$. The product: $threadsNumber * memsize$ should be equal to the size of A. At the beginning of each run, every thread copies the assigned part of the array A from the global memory to its local registers, then calculates number of 0 and 1 bits. At the end of the run, the content of the global memory array is rearranged – each thread moves the elements from its part of A.

Each of the three implementations of the radix sort algorithm was tested on a randomly generated array of 65536 integers, with five combinations of $threadsNumber$ and $memsize$ parameters. For the given implementation and the value of parameters, the numbers of arithmetic operations and accesses to the global and shared memory were calculated and used in the PN model (as the initial marking of the places p_5, p_7 and p_9). As an arithmetic operation we count every assignment, addition, subtraction, multiplication, division, relational operation, logical operation and array subscript (arrays in registers). Every access (read or write) to data stored in the global or shared memory is counted as single memory operation.

The numbers of steps of the model calculations were compared to the execution times of kernels. The tests were performed on NVIDIA GeForce GTX 960M graphical card with CUDA Toolkit 8.0. The execution times of kernels are averages of one hundred runs. The results for the PN model were calculated as averages of ten computations of the models.

In the first implementation only the global memory is used. The second phase of the algorithm is performed by thread with id *0*. The results of the tests are depicted in Figure 4 – the dotted line.

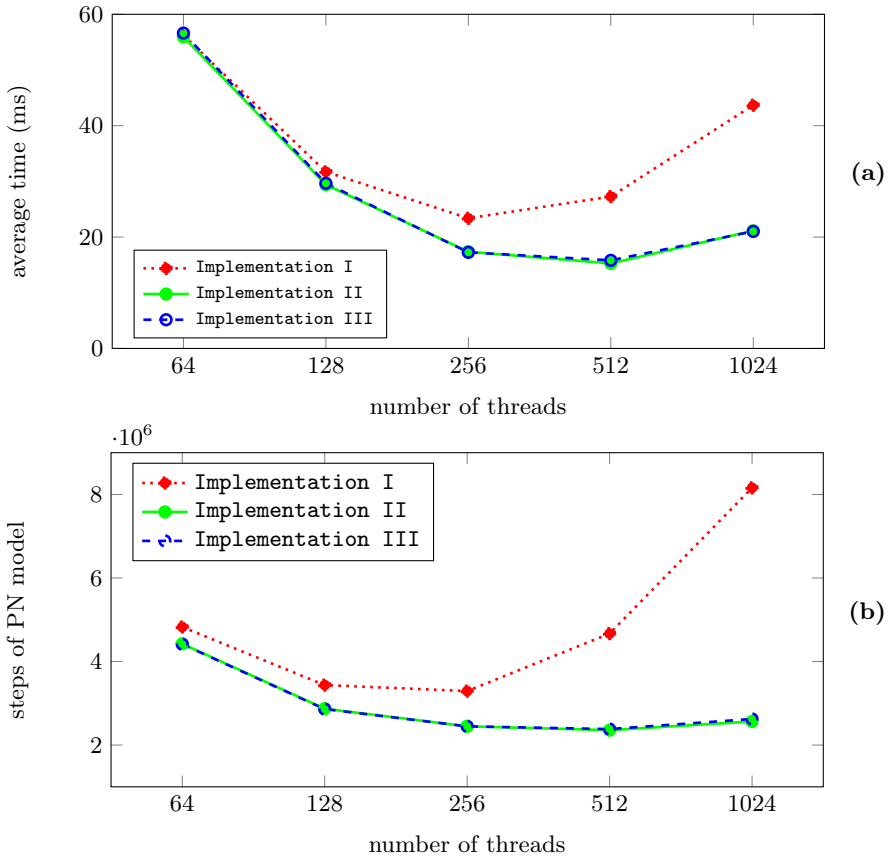


Fig. 4. The results for the radix sort algorithm tests with 65536 element arrays: (a) execution times of kernels (ms), (b) steps of the PN model.

In the second implementation the realization of the second phase is organized in a more efficient way. Instead of computing all sums incrementally, we compute all partial sums (for indexes between $p \cdot 2^q$ and $(p+1) \cdot 2^q$, where p and q are

suitable non-negative integers) and use them as input components for other sums.

Having $m = 2^r$ nodes we can compute $zeros[i, j]$ and $ones[i, j]$ for all $j \leq m$ in $r+1$ cycles with full system load (using all nodes in every cycle). To do it, we compute in $c-th$ cycle specific partial sums of lengths between 2^{c-1} and $2^c - 1$, an example for $r = 2$ is depicted on Figure 5. In this sample case $z[x..y]$ denotes $\sum_{x \leq t \leq y} zeros[i, t]$, while $o[x..y]$ – $\sum_{x \leq t \leq y} ones[i, t]$, each row corresponds to a single element in table *zeros* or *ones*, while in subsequent columns the values of partial sums stored in those elements are given. Each arc between $x - th$ and $y - th$ row denotes the addition of value kept in $x - th$ element to the value kept in $y - th$ element, the result is stored in $y - th$ element.

More specifically, in the first cycle we compute

$$zeros[i, 2k] + zeros[i, 2k + 1] \quad \text{and} \quad ones[i, 2k] + ones[i, 2k + 1]$$

placing results in $zeros[i, 2k+1]$ and $ones[i, 2k+1]$ respectively. The number of all operations made in this cycle (the number of arcs between first and second column in example) is $m/2 + m/2 = m$, hence we can utilize all available nodes to do it at once.

In subsequent cycles we compute longer partial sums using already precomputed ones. This way in $c - th$ cycle we compute

$$\sum_{0 \leq t \leq u} zeros[i, 2^c k + t] \quad \text{and} \quad \sum_{0 \leq t \leq u} ones[i, 2^c k + t],$$

where $2^{c-1} \leq u < 2^c - 1$, while $0 \leq k < 2^{r-c} - 1$, and store the results in $zeros[i, 2^c k+t]$ and $ones[i, 2^c k+t]$, respectively. Since after the previous cycle all values $\sum_{0 \leq t \leq u} zeros[i, 2^c k+t]$ and $\sum_{0 \leq t \leq u} ones[i, 2^c k+t]$ are stored in memory, where $2^{c-2} \leq u < 2^{c-1}$, while $0 \leq k < 2^{r-c+1}$, we need to add only two elements for each longer partial sum computed in $c - th$ cycle. Note that the number of such operations equals to the size of the range of u multiplied by the size of the range of k and doubled (we need to compute both ones and zeros), i.e.

$$|\{u, 2^{c-1} \leq u < 2^c\}| \cdot |\{k, 0 \leq k < 2^{r-c}\}| \cdot 2 = 2^{c-1} \cdot 2^{r-c} \cdot 2 = 2^r = m.$$

Finally, in the last cycle we add the computed so far $\sum_{k \leq m} zeros[i, k]$ to all $\sum_{t \leq k} ones[i, t]$ for each $k \leq m$. The execution times of kernels and the results from the PN model are depicted in Figure 4 – the solid line.

In the third version of the implementation the arrays *zeros* and *ones* are stored in the shared memory instead of global. The results of those tests are also presented in Figure 4 – the dashed line.

The results of the PN model calculations for the first implementation (Figure 4 (a)) clearly show that this parallelization is not very efficient as compared to the others, especially for larger numbers of threads. This is confirmed by the execution times of kernels (Figure 4 (b)). One can easily observe that in both plots the number of PN model steps and execution times of kernels for this implementation initially decrease with the increasing number of threads, however

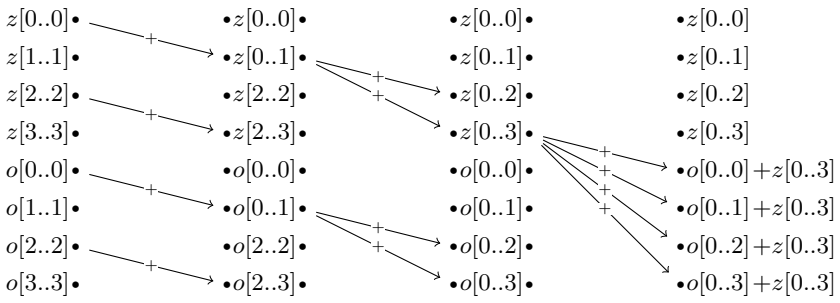


Fig. 5. The organization of the second phase of the algorithm for $m = 4$ (as $m = 2^r$, $r = 2$) nodes in $r + 1 = 3$ cycles.

for more than 256 threads both of them increase. The similar situation can be also noticed for other implementations, but here the growth is more significant. It can also be observed that for the largest amounts of threads the number of PN model steps increases faster than execution times of kernels. In this case the general direction of changes predicted by our model is consistent with the kernels executions (despite of the lack of the exact match of the plots). The model predicted correctly the most efficient choice of the number of threads, which in this case is 256.

The predictions of the PN model for the 2nd and 3rd implementations are more consistent with the execution times of kernels. In both cases differences between implementations are very small. It is probably caused by a relatively small number of shared memory operations in comparison to accesses to the global memory. For the larger number of threads, the increase in execution times of kernels is more significant than in the results from the PN model. The reliable explanation of such a difference is an overhead for communication between threads. It is clear that such overhead will not be observed in the PN model. However, the general characterization of the results is the same both for the model and the kernels. The PN model predicted correctly also the most effective choice of the number of threads for both implementations, which is 512 threads.

6 Conclusions and future work

The purpose of our PN based GPU computations and memory access model is to help in the analysis and optimization parallel algorithms, which are designed to be implemented on CUDA graphical cards. We do not require the source code to be given as an input, however the algorithm description should be detailed enough to estimate the number of arithmetic operations and accesses to the global and shared memory. Note that the other tools (e.g. the roofline model) also require those information. The expected speedup of the computation from a parallelization can be predicted and compared to other algorithms, even with-

out using of any physical GPU device. The model can help to predict which algorithm is the fastest, how much its modifications can affect the speedup of the computation and whether they are significant enough to include them in the source code.

Any inaccurate results demonstrated by our model might be interpreted as a premise that the algorithm should be improved. As an example recall the presented results for the radix sort algorithm. The predictions of the PN model for the first implementation were not satisfying, and the model clearly showed that a different organization of the second phase of the algorithm improved the time of the computation. On the other hand, using the shared memory in this case was not so beneficial. That was confirmed by the GPU kernels execution times.

Another important advantage of the PN model is the possibility to check how different values of parameters and the level of parallelization can affect the final efficiency. As it can be observed in Figure 4, it is not only a theoretical discussion, the problem is substantial and can result in very different execution times of kernels. With the appropriate number of threads, the GPU calculations were even three times faster. It should be also noticed that for all three implementations presented above, different numbers of threads were the most efficient, hence the selection of the one, universal number of threads is not possible. Our model easily allows to check different number of threads (warps) and its predictions seem to be very accurate. Various values of parameters may also result in different numbers of arithmetic operations and accesses to the memory, and it can be also easily introduced and analyzed by the model.

Our model can freely swap instructions of various types. It allows to check whether different order of the instructions may improve the algorithm efficiency, for example by allowing to hide thread waiting periods in calculations. If the results of the PN model are significantly better than the results from GPU kernels execution, that possibility should be considered. Naturally, the swap of the instructions is not always possible because of the nature of calculations. One of the most important improvements of the model would be the introduction of a partial order over the set of instructions. This can be achieved by defining dependence of instructions basing on the access to the same variable (see [9]). The partial order would make predictions of the model more accurate.

In the model, the bounded inhibitor Petri nets are used. In the future we would like also to consider other PN semantics like colored PNs, timed PNs, etc.. Their nature seems to be suitable for our model and they may help to simplify it or introduce new features.

The PN results are expressed as the number of steps performed, while for the GPU computations we use the execution times of kernels in milliseconds. To compare them directly the special coefficient is required to align one result with the other. However, different nature of various algorithms as well as the lack of research on atomic and comparable in terms of time consumption operations makes the issue of finding the universal coefficient a very hard task.

Although designing of the efficient sorting algorithm was not the aim of this paper, we described the process of improving the parallel version of the considered radix sort algorithm. Nevertheless, it is worth to note that providing its further improvements is possible. The radix sort is quite popular, both in the most significant digit (MSD), normally together with merge sort subroutine [5], and the least significant digit (LSD), as suggested in [10], version.

Note that for different GPU devices the execution time of a fixed kernel may differ, while the number of steps performed by the model stays the same. It would be useful to prepare a set of benchmarks, which for a given device compute the universal scaling coefficient for this device and the model.

References

1. Giovanni Chiola, Susanna Donatelli, and Giuliana Franceschinis. Priorities, inhibitor arcs and concurrency in p/t nets. In *Proc. of ICATPN*, volume 91, pages 182–205, 1991.
2. Nvidia Corporation. CUDA. `http://www.nvidia.com/object/cuda_home_new.html`.
3. Nvidia Corporation. CUDA. Best practice guide version 8.0.61, 2017.
4. Nvidia Corporation. CUDA. C programming guide version 7.5, 2017.
5. Victor J. Duvanenko. Algorithm improvement through performance measurement. `http://www.drdobbs.com/architecture-and-design/algorithm-improvement-through-performanc/220000504`, 2009.
6. Ananth Grama. *Introduction to parallel computing.* Pearson Education, 2003.
7. Sunpyo Hong and Hyesoon Kim. An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness. In *ACM SIGARCH Computer Architecture News*, volume 37, pages 152–163. ACM, 2009.
8. Kai Hwang and Naresh Jotwani. *Advanced Computer Architecture, 3e.* McGraw-Hill Education, 2011.
9. Ryszard Janicki and Maciej Koutny. Structure of concurrency. *Theoretical Computer Science*, 112(1):5–52, 1993.
10. David Luebke, John Owens, Mike Roberts, and Cheng-Han Lee. Intro to parallel programming – online course. Nvidia Corporation.
11. Lin Ma, Kunal Agrawal, and Roger D Chamberlain. A memory access model for highly-threaded many-core architectures. *Future Generation Computer Systems*, 30:202–215, 2014.
12. Lin Ma, Roger D Chamberlain, and Kunal Agrawal. Performance modeling for highly-threaded many-core gpus. In *Application-specific Systems, Architectures and Processors (ASAP), 2014 IEEE 25th International Conference on*, pages 84–91. IEEE, 2014.
13. Rishabh Mukherjee. *A Performance Prediction Model for the CUDA GPGPU Platform.* PhD thesis, International Institute of Information Technology Hyderabad, India, 2010.
14. Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.
15. David A Patterson. *Computer architecture: a quantitative approach.* Elsevier, 2011.
16. Wolfgang Reisig. *Petri nets: an introduction*, volume 4. Springer Science & Business Media, 2012.

17. Nadathur Satish, Mark Harris, and Michael Garland. Designing efficient sorting algorithms for manycore GPUs. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–10. IEEE, 2009.
18. Harold E Seward. Information sorting in the application of electronic digital computers to business operations, Master Thesis, MIT, 1954.
19. S Shuaiwen, S Chunyi, R Barry, and C Kirk. A simplified and accurate model of power-performance efficiency on emergent gpu architecure. In *Parallel &Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 673–686, 2013.
20. Duane Storti and Mete Yurtoglu. *CUDA for Engineers: An Introduction to High-performance Parallel Computing.* Addison-Wesley Professional, 2015.
21. Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

Coloured Petri Nets Based Diagnosis on Causal Models

Soumia Mancer and Hammadi Bennoui

Computer science department, LINFI Lab.
University of Biskra, Algeria
mancer.soumia@gmail.com, bennoui@gmail.com

Abstract. In the last decades, several approaches have been proposed in order to capture the problem of causal model-based diagnosis within Petri Nets (PNs) framework, where both the structural and behavioural analysis of the net model are exploited for reasoning. In fact, PNs are a useful tool, but most of the approaches suffer from the large size of the obtained models even for simple systems. This paper introduces a novel class of Coloured Petri Nets (CPNs) called Causal CPNs. Such a net model is motivated by representing the causal behaviour of the system to be diagnosed, as well as, simplifying the analysis methods. The diagnosis technique exploits backwardly the reachability graph of the net model. A case study is used to illustrate the usefulness of our proposal for fault diagnosis.

Keywords: model-based diagnosis, causal models, coloured Petri nets, reachability graphs

1 Introduction

Model-based approach as an alternative to heuristic based one, especially when the experimentations are missing, deals widely with fault diagnosis in such a manner that the examination of a given system is done on the basis of a model. It aims at explaining any observed behaviour that conflicts with the way the system is meant to behave. Among the diagnosis frameworks found in the literature, those based on causal models where the explanations would be given in terms of initial causes leading the system to a misbehaviour.

In logical frameworks, a causal model-based diagnosis problem is traditionally solved through symbolic manipulations that are shown as a cumbersome task; and hence, numerous attempts to face this problem have been done. In particular, Petri nets (PNs) have been used to represent the causal model, and so to exploit their analysis techniques to implement efficiently the diagnosis reasoning mechanisms [1, 2]. For problems where the net model is large or composed of some identical parts, it is well known that Coloured PNs (CPNs) are well suited to use with respect to classical PNs. By means of data type primitives, it is possible to achieve a reduced model about the behaviour of the system under examination. The manipulation of the data values carried by tokens that

reside in places of a CPN model is done through the arc expressions. Generally, those expressions are functions that define the added/removed tokens to/from a place. Nevertheless, when analysing the CPN model backwardly, those expressions may exhibit a more complicated process because of the inversion task. In order to simplify this latter, and so, the analysis phase, we propose resorting to matrices as a way of manipulating the token colours in the net model.

In this paper, we focus on the use of CPNs for causal model-based diagnosis. For that reason, we introduce a particular CPN called Causal-CPN (CCPN) that allows representing the causal behaviour of the system to be diagnosed by means of causal matrices that are attached to transitions of the net model. Causal matrices are used to define the possible inputs and their associated outputs of transitions as causal relationships. For solving a given diagnosis problem based on CCPN, a backward analysis on the corresponding markings graph is defined in this paper to generate the possible diagnoses. In fact, such analysis can be seen as the coloured version of the BW-analysis that has been proposed in [2] for some simplified Petri nets called Behavioural Petri Nets.

The present paper starts in section 2 by outlining briefly some basic definitions on which we will rely throughout the paper. Section 3 introduces the CCPN model by which it is possible to restrict CPNs for representing causal models. A formalisation of the model-based diagnosis problem by means of CCPNs is detailed in section 4, and solving such a problem is shown in section 5 by exploiting the backward reachability analysis on reachable markings graph. Finally, section 7 concludes the paper and outlines future work.

2 Preliminaries

In this section, we outline some basic definitions that we need in this paper.

2.1 Causal model

From [1], a causal model is a couple (V, E) where V is a set of entities, noted states, and E is a set of cause-effect relationships among states. For the diagnosis reason, states are classified into Initial-causes, Internal states and Manifestations. The states of a causal model are used to represent the states (partial states) of the modelled system. Initial causes represent initial states from which any evolution in the system begins. Internal states describe the unobservable part of the system as consequences of initial states. Manifestations represent the observable states of the system as consequences of internal states. Each of the states can be instantiated by assigning a value to it from a finite and predefined set noted $admissible\_values$. It is important to keep in mind that each state must assume at most one value at a given time and it can be present on the model or absent.

2.2 diagnosis problem

A diagnosis problem is defined logically in [3] as a triple $DP = (BM, INIT, < \Psi^+, \Psi^- >)$ where BM is the behavioral model of the system to be diagnosed,

$INIT$ is the set of instances of initial causes in terms of which the observations have to be explained, Ψ^+ is a subset of observations to be entailed by a solution of DP and Ψ^- is the set of all possible values that conflict with the made observation (that are known to be absent in the case under examination). Let OBS be the current set of observations, thus, $\Psi^+ \subseteq OBS$ and $\Psi^- = \{m(x)|m(y) \in OBS, x \neq y\}$ such that m is a manifestation and $x, y \in admissible\_values(m)$. A solution to DP is a set $\Delta \subseteq INIT$ such that Δ predicts each parameter in Ψ^+ and no parameter in Ψ^-:

$$\begin{array}{ll} \forall x \in \Psi^+ & BM \cup \Delta \vdash x \\ \forall y \in \Psi^- & BM \cup \Delta \nvdash y \end{array}$$

Where $\vdash$ is the derivation symbol.

2.3 Coloured Petri nets

Coloured Petri Nets [4] are high-level nets merging both Petri Nets and the functional programming language Standard ML in one model. CPNs still retain, as strong points of PNs, the foundation of the graphical notation and the basic primitives for modelling concurrency, communication and synchronisation, while Standard ML provides the primitives for data types definition and data values manipulation.

As CPNs are basically PNs, it is important to keep in mind that a CPN model is defined by a finite set of places, a finite set of transitions and a finite set of arcs as connections between places and transitions. Each place has an associated type and may hold one or more tokens each of which carries a data value belonging to the place's type. By convention, types are colour sets thus, tokens are token colours.

Each place has its own marking which is a multiset of token colours that are present in such a place. The sum of individual place markings gives the marking of the CPN model. The marking changes during the execution of the model by means of transition's firing. As it is known, when a transition occurs it removes tokens from its input places and it adds tokens to its output places. In CPN, the removed (resp. added) tokens are determined by an arc expression associated to the outgoing (resp. incoming) arc from (resp. to) a place. An arc expression is built from typed variables, constants, operators, and functions and it evaluates to a multi-set of token colours. Moreover, it can be attached to each transition a boolean expression (with variables) called a guard which specifies the bindings for which it evaluates to true. A binding is an assignment of data values to the free variables appearing in the expression of an incoming arc or a guard of a transition. A binding of a transition can be written in the form: $(v_1 = d_1, v_2 = d_2, ..., v_n = d_n)$ where $for\ i \in 1..n : v_i$ is a variable and d_i is the value assigned to v_i.

A transition t is enabled, ready to occur, if there is a binding such that: 1) the evaluation result of each of the input arc expressions is present on the corresponding input place; and 2) the guard (if any) is satisfied. The firing of a

t adds to each output place a multi-set of token colours to which the expression on the corresponding output arc is evaluated.

Before recalling the formal definition of a CPN from [4], we recall the concept of *multi-sets*. A multiset is a set where individual elements may occur more than once.

Definition 1. *(**Multi set**) A multiset m, over a non-empty set S, is a function $m \in [S \to \mathbb{N}]$ represented formally by: $\sum_{s \in S} m(s)'s$.*

- *$\forall s \in S : s \in m$ iff $m(s) \neq 0$.*
- *S_{MS} is set of all finite multi-sets over S.*

A set of operations that can be applied on multisets are defined as follows: $\forall m, m_1, m_2 \in S_{MS}$ and $\forall n \in \mathbb{N}$:

$$\begin{aligned} |m| &= \textstyle\sum_{s \in S} m(s). \\ m_1 + m_2 &= \textstyle\sum_{s \in S} (m_1(s) + m_2(s))'s. \\ m_1 \neq m_2 &= \exists s \in S : m_1(s) \neq m_2(s). \\ m_1 \leqslant m_2 &= \forall s \in S : m_1(s) \leqslant m_2(s) \\ & \quad \textit{(defined analogously for } \geqslant\textit{).} \end{aligned}$$

Thus, a CPN is given by:

Definition 2. *A Coloured Petri net (CPN) is a 6-tuple $N = (\Sigma, P, T, A, C, G)$ where:*

- *Σ is a non-empty set of types (colour sets).*
- *P is a non-empty set of places.*
- *T is a non-empty set of transitions.*
- *$P \cap T = \emptyset$, $P \cup T \neq \emptyset$.*
- *A is a non-empty set of arcs such that: $A \subseteq (P \times T) \cup (T \times P)$.*
- *$C : P \to \Sigma$ is a colour function maps each place into a colour set.*
- *G is a guard function maps each transition into a boolean expression.*

Definition 3. *A marked CPN is a pair (N, μ) where N is a CPN and μ is a function defined on P such that:*

$$\mu(p) \in C(p)_{MS}, \forall p \in P.$$

3 Causal-Coloured Petri nets

CPNs are used for representing and analysing a large variety of systems. The colour set concept allows us to obtain a reduced net model. While through the expressions associated with transitions (guards) or the arcs we define the possible combinations of input and output token colours for a transition to fire, and so, describing the evolution of the system under study. All the analysis techniques defined for classical PNs are extended for CPNs. Among them, we are interested in backward ones. The backward analysis of a CPN seems as an inversion of

the net model [5] and hence realising the analysis forwardly. The main problem that arises, here, is the backfiring of a transition. When inverting the arcs, their expressions have to be inverted too, and the same as the transition's guard. Such an inversion depends on the input and the output arc expressions of the transition. It may lead to a combinatorial explosion in some cases. In order to face this problem, we propose associating with each transition a matrix describing the possible combinations of input and output token colours. As a result, the expressions of the input arcs of a transition are typed variables and the output ones are simple expressions that extract from a given matrix and according to the input tokens the possible output ones. In the following, we introduce a particular class of CPNs called Causal CPNs characterised by matrices associated with their transitions. CCPNs are used to represent the causal behaviour of the system under examination.

Definition 4. *A Causal-Coloured Petri Net is a 6-tuple* $N = (\Sigma, P, T, A, C, FW)$ *where:*

- (Σ, P, T, A, C) *is a CPN.*
- $P = Ic \uplus Is \uplus Mn$*:*
  $Ic = \{p | p \in P,\ {}^{\bullet}p = \emptyset\}$[1]*,* $Mn \subseteq \{p | p \in P,\ p^{\bullet} = \emptyset\}$ *while* $Is = P \setminus (Ic \cup Mn)$*.*
- A^{+}*, denotes the transitive closure of* A*, is irreflexive.*
- $FW : T \longrightarrow MAT_{n,m}(\bigcup_{\omega \in \Sigma} \omega)$[2] *such that:*
  - n *is the number of transitions for which a transition* t *may be unfolded in a classical PN (i.e, it is the number of ways that* t *may fire.)*
  - $m = |{}^{\bullet}t| + |t^{\bullet}|$ *defines the number of places connected with* t*, either inputs or outputs.*

Definition 5. *A marked CCPN is a pair* (N, μ) *where* N *is a CCPN and* μ *is a marking such that:*

$$\forall p \in P :\ |\mu(p)| \leqslant 1.$$

Let μ_0 *be the initial marking of* N *such that:* $\forall p \in P :\ \mu_0(p) \neq \emptyset \rightarrow p \in Ic$*.*

Definition 6. *A marked CCPN* (N, μ_0) *is said to be safe iff:*

$$\forall p \in P,\ \forall \mu \in R(N, \mu_0) : |\mu(p)| \leqslant 1.$$

Where $R(N, \mu_0)$ *denotes the reachability set from* μ_0*.*

Let us introduce the following notations for the rest of the paper.

Note 1. $\forall t \in T$ and $\forall (x_1, x_2) \in A$:

– $A(t)$ denotes the set of the input arcs of t.

[1] ${}^{\bullet}x = \{y | (y, x) \in A\}$ and $x^{\bullet} = \{y | (x, y) \in A\}$ denote the input and output sets of x respec. where $x \in P \cup T$.

[2] $MAT_{n,m}(\bigcup_{\omega \in \Sigma} \omega)$ defines the space of matrices of n lines and m colomns.

- $Var(t)$ denotes the set of variables that appear in the input arc expressions of t and in its associated guard $G(t)$ (it is also used for expressions $Var(expr)$). In a CCPN model: $Var(G(t)) = Var(t)$.
- $E(x_1, x_2)$ denotes the expression attached to an arc (x_1, x_2).

Definition 7. *A binding of a transition t is a function b defined on $Var(t)$ such that:*

- $\forall v \in Var(t) : b(v) \in Type(v)$.
- *Let $Var(t) = \{v_1, ..., v_n\}$: $[b(v_1)\ b(v_2)\ ...\ b(v_n)]$ be the ith sub-vector-line of $FW(t)$.*

$Expr < b >$ denotes the evaluation of the expression $Expr$ in the binding b.

As a particularity of CCPN, in comparison with CPN, is the irreflexivity of A^+; hence, the CCPN model is acyclic. Thus, it is possible to introduce a partial order, noted $\prec$, between its transitions. Such an order is inspired from that which is defined for BPNs in [2] as follows:

$$\text{Let } t_1, t_2 \in T : t_1 \prec t_2 \Leftrightarrow t_1 A^+ t_2.$$

Definition 8. *Let μ be a marking, a transition t is enabled at μ iff:*

$$\forall p \in {}^\bullet t : E(p,t) < b > \leqslant \mu(p) \text{ and } \nexists t' \prec t \text{ s.t } t' \text{ is enabled.}$$

Definition 9. *Let t be an enabled transition in a marking μ, the firing of t changes the marking μ to anothor marking μ' as follows:*

$$\forall p \in P : \mu'(p) = \mu(p) - E(p,t) < b > + E(t,p) < b > .$$

Definition 10. *Given a marked CCPN (N, μ), we denote by a step the set of enabled and concurrent transitions at μ.*

Definition 11. *Given a marked CCPN (N, μ) and a step $s = \{t_1, .., t_n\}$, the firing of s at μ reaches the new marking μ' such that $\mu' = \bigcup_{i=1}^{n} \mu_i, \mu[t_i > \mu_i$.*

In a CCPN model, places are used to represent the entities of the causal model of the system to be diagnosed. As we have mentioned, it suffices to distinguish, for diagnosis purposes, among three classes of entities: *Initial causes*, noted Ic, represented by source places, *Internal states*, noted Is, represented by places that have input transitions, and *Manifestations*, noted Mn, represented by sink places. Transitions represent the cause-effect relations among corresponding places. To each transition is attached a matrix given by $FW(t)$ for which its lines represent the different ways that such a transition t can fire while each of its columns is associated with a place p that surrounds t, and so, it determines the possible values that will be consumed/produced from/in p by the firing of t. Each transition matrix $FW(t)$ can be divided into two sub-matrices $FW\_in(t)$ and $FW\_out(t)$. FW_in corresponds to the possible combining token colour

inputs of t while FW_out is the output sub-matrix of t. Furthermore, each transition t can be classified as joint or fork transition, it is noticed that the linear transition is a particular fork or joint transition. A joint transition is used, as usual, to represent a conjunction of places, but also, it is used, in a CCPN, to deal with both cases where there are several concurrent possibilities for reaching a place or where there is an exclusive-or between, at least, two execution paths for reaching that place. As the marking of a place and the evaluation of an arc expression is a multiset, we inspire the idea of exploiting the empty sets as components of a joint transition matrix to describe the mentioned cases above for better and more logical representation of the system behaviour. In the case where there are several concurrent evolutions starting from a place by the same value, a fork transition will be used to duplicate the required place by the specified marking for each path. For each transition t, the input arc expressions are typed variables while the output arc expression is a function that holds as inputs a binding vector and the transition matrix and returns the output token colour that corresponds to such a binding. The net model is safe, that is, any place must be marked by, at most, one token colour.

Definition 12. *Let (N, μ) be a marked CCPN, the marking μ may lead to an inconsistency iff:* $\exists t_i, t_j \in T, \exists \mu_i, \mu_j \in R(N, \mu)$:

$$\mu[t_i > \mu_i \wedge \mu[t_j > \mu_j \wedge \exists p \in P | \mu_i(p) \neq \mu_j(p).$$

Example 1. As an example of a CCPN model, we consider a simple model adapted from an example given in [1] which is used to represent a partial fault model of a car engine. However, it is representative enough for introducing the basic constructs of CCPNs. Fig.1 gives the graphical representation of the considered model. Such a model is characterized by c_1, c_2 and c_3 as initial-causes of the described causal model, and m_1 and m_2 as manifestations. The left places represent the internal states. Each place has a type, attached to it, which determines the set of colours that the token on the place is allowed to have. As a sample, the tokens residing in c_1 will have an a or b as their token colour.

The transitions are used to model the cause-effect relationships among the corresponding entities in the causal model. Each transition is labelled by a matrix that defines its firing ways. t_1 is a fork transition that is used to duplicate the input place c_1 by the marking a into c_{11} and c_{12} by the same token color. In $FW(t_1)$, the first column corresponds to the input place c_1, while, the last two ones correspond to the output places c_{11} and c_{12}. t_2 will be enabled only if one of the input places is marked by the colour a, and as a result, it produces an a or b colour in s_1. The transition t_3 is as the logical *and* while t_5 and t_8 are as the logical *or*. The transition t_8 can be enabled in three cases (according to the number of lines in its matrix), the two first ones represent the case when either s_3 or s_4 is marked by a color a while the last is that when both places are marked by such a color. The transition t_5 has the same interpretation as t_8, but only, with t_8 the produced color is the same for all the cases (firing ways), and so, it is used to guard the safeness of the model (each place is marked at most by one color at a given time), while with t_5 the produced color is defined according

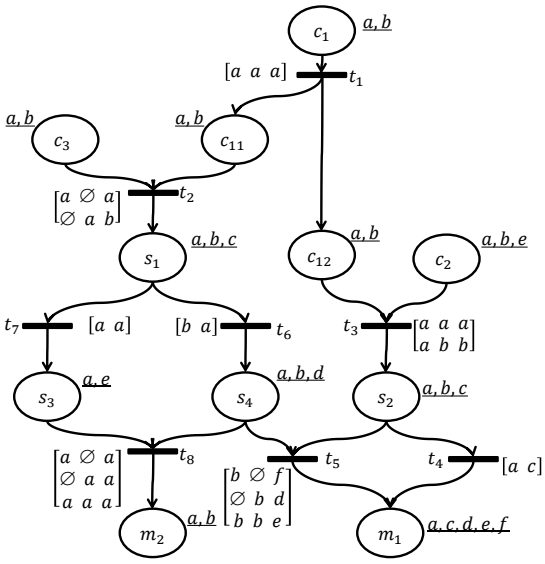


Fig. 1. A CCPN example.

to the presence of colors in s_2 and s_4, and here, t_5 guards the consistency of the net model.

4 Formalising diagnosis with CCPNs

CCPNs, as particular CPNs for representing the causal behaviour of a system, are introduced mainly to deal with fault diagnosis. In this section, we show how the diagnosis problem that is given in Definition 2.2 can be formalised in the basis of a CCPN model. A diagnosis problem is pointed out when it appears a discrepancy between the required behaviour of the examined system and its real one. In causal model-based diagnosis, the observed behaviour is given as a set of manifestations, noted OBS. In CCPNs, it consists of a final marking μ^{OBS} where the marked places are those belonging to Mn. Formally, it is given by:

Definition 13. *Given a CCPN as a model of the system S, an observation is a marking μ^{OBS} such that:*

$$\forall p \in P : \mu^{OBS}(p) \neq \emptyset \rightarrow p \in Mn.$$

Definition 14. *A diagnosis problem is defined in terms of a CCPN model by the triple: $DP = (N, INIT, < M^+, M^- >)$ where:*

- *N is the CCPN model.*
- *$INIT = \{(p, c) | p \in Ic, c \in C(p)\}$.*

- $M^+ = \{(p,c)|p \in Mn, c \in C(p), \mu^{OBS}(p) = c\}$.
- $M^- = \{(p,c)|p \in Mn, c \in C(p), \mu^{OBS}(p) \neq c\}$.

In this definition, N represents the causal behavioural model of the system to be diagnosed. $INIT$ is a set of couples (p,c) in terms of which diagnosis solutions would be given. It consists of a set of possible markings c of each place p that represents an initial cause in the causal model. $< M^+, M^- >$ represents the made observation. M^+ consists of a set of couples (p,c) representing manifestations that have to be entailed by a solution to DP, while M^- is the set of couples (p',c') that conflict with the made observation (*i.e*, it is used to ensure the required consistency).

Definition 15. *Let N be a CCPN, $p \in P$, $c \in C(p)$ and μ_0 an initial marking;*

$$(N,\mu_0) \vdash (p,c) \leftrightarrow \exists \mu \in R(N,\mu_0)|\mu(p) = c.$$

While,

$$(N,\mu_0) \nvdash (p,c) \leftrightarrow \forall \mu \in R(N,\mu_0)|\mu(p) \neq c.$$

Definition 16. *Let N be a CCPN, $p \in P$, $c \in C(p)$ and μ_0 an initial marking; A generalization of Definition 15 for a set $Q = \{(p,c)|p \in P, c \in C(p)\}$ is given by:*

$$(N,\mu_0) \vdash Q \leftrightarrow \exists \mu \in R(N,\mu_0)|\forall (p,c) \in Q : \mu(p) = c.$$

While,

$$(N,\mu_0) \nvdash Q \leftrightarrow \forall \mu \in R(N,\mu_0)|\forall (p,c) \in Q : \mu(p) \neq c.$$

The notion of diagnosis solution can be now captured by the following proposition.

Proposition 1. *Given a diagnosis problem $DP = (N, INIT, < M^+, M^- >)$, an initial marking μ_0 is a solution to DP iff:*

$$(N,\mu_0) \vdash M^+ \quad \textit{and} \quad (N,\mu_0) \nvdash M^-.$$

This means that μ_0 has to account for all observations in M^+, while, no one in M^- must be reached from μ_0.

Proof. We proceed to prove this proposition by contradiction.

($\rightarrow$)
Suppose that μ_0 is a solution to DP and that $[(N,\mu_0) \vdash M^+ \wedge (N,\mu_0) \nvdash M^-]$ does not hold.
By Definition 16: $(\forall \mu \in R(N,\mu_0) \exists (p,c) \in M^+ : \mu(p) \neq c) \vee (\exists \mu \in R(N,\mu_0) \exists (p,c) \in M^- : \mu(p) = c)$.
And so, μ_0 is not a solution, which contradicts with our first assumption.

($\leftarrow$)
Suppose that $[(N,\mu_0) \vdash M^+ \wedge (N,\mu_0) \nvdash M^-]$ and that μ_0 is not a solution.
By Definition 16: $(\exists \mu \in R(N,\mu_0)|\forall (p,c) \in M^+ : \mu(p) = c) \wedge (\forall \mu \in R(N,\mu_0)|\forall (p,c) \in M^- : \mu(p) \neq c)$.
As a result, μ_0 is a solution, which is a contradiction.

□

5 diagnosis problem solving with CCPNs

In order to solve a diagnosis problem given by means of a CCPN, we are interested in the backward reachability analysis. In this section, a formalisation of the method is given for analysing a CCPN model, and so, generating a set of possible diagnoses that explain a given malfunction.

Generally, a backward analysis on reachability graphs allows determining the markings from which a given marking is reachable. It can be done by a simple direction inversion of the arcs in classical PNs, while, the transition's firing rule remains the same as forwarding one. When dealing with CPNs, the inversion has to be applied on arc expressions and transition guards that are, generally, functions. For linear transitions, the inversion process is trivial. For the case of a fork or joint transitions, the inversion process becomes complicated. Thus, it is possible to fall in a combinatorial explosion problems. In a CCPNs, in addition to direction inversion of the arcs, the inversion process can be achieved simply by a re-ordering of the transition matrices FW. In such a way, the input submatrix FW_in becomes output, denoted b_FW_out. While, the output submatrix FW_out becomes input, denoted b_FW_in. As a result, FW becomes b_FW. And so, arc expressions have to be changed. As usual, each input arc expression of a transition t is a typed variable; while, each output one equals to the following expression $f([v_{q_1}..v_{q_m}], b\_FW(t))(p)$ such that, p is added to specify which place is ($i.e$, $f([v_{q_1}..v_{q_m}], b\_FW(t))$ is a vector, for which, each of its components corresponds, backwardly, to an output place).

Definition 17. *Let M be a marking, a transition t is backwardly enabled in M iff:*

$$\forall p \in \ t^{\bullet} : E(t,p) < b > \leqslant M(p) \text{ and } \nexists t' \succ t.$$

Such that t' is backwardly enabled and the binding b is defined over $b\_FW(t)$.

Assuming that a transition t is backwardly enabled, the backfiring of t makes the execution process of the model returning back, where it removes token colors $\{c_i\}_{1 \leq i \leq n}$ from the output places of t and adds a set of others $\{c'_j\}_{1 \leq j \leq m}$ each of which to its own place that belongs to the input ones of t. This set of colours can be easily defined by the backward transition matrix of t. By the same manner as forwarding one, the new marking μ' can be calculated from μ after firing the transition t. Furthermore, μ may lead to an inconsistent marking (Definition 12). A marking μ is an inconsistent one for the case when there is a fork transition t in which its output places have different markings other than the empty set.

Definition 18. *Let (N, μ) be a marked CCPN, μ is said to be inconsistent iff:*

$$\exists t \in T, \ \exists p, p' \in t^{\bullet} : \mu(p) \neq \mu(p').$$

Definition 19. *Given a marked CCPN (N, μ), let t be a fork transition such that ${}^{\bullet}t = \{p\}$ and $t^{\bullet} = \{p_1, ..., p_m\}$, t is forced backwardly at the marking μ iff:*

- *t is not backwardly enabled at μ.*

- $\exists p_i (1 \leq i \leq m) | \mu(p_i) \neq \emptyset$.
- *μ is not inconsistent.*
- *$\nexists t' \succ t$ where t' is backwardly enabled or forced at μ.*

If t is forced at μ then $\forall p, p' \in t^\bullet$ such that $\mu(p) = \emptyset$ and $\mu(p') \neq \emptyset$, consider $\mu(p) = \mu(p')$.

Solving a diagnosis problem given by DP consists of constructing backwardly a reachability graph. We start such a construction from a submarking μ of μ^{OBS} such that $\forall (p, c) \in M^+ : \mu(p) = c$, while the others are empty. The terminal nodes of the graph can be initial markings ranged in a set μ^{ini}, inconsistent markings or markings leading to an inconsistency. The arcs of the graph are labelled by steps (It should be noted that, in this case, a step is a set of backwardly enabled transitions at the given marking). The set of initial markings μ^{ini} is a set of markings μ_i in such a manner that μ is a submarking of μ' where $\mu' \in R(N, \mu_i)$. The set μ^{ini} represents the candidate solutions to the given problem, thus, a consistent solution is a candidate marking that does not, by any way, lead to any combination in M^-. To ensure that, we build for each marking of μ^{ini} the corresponding forward reachability graph. We select as consistent solutions the markings that reach no element of M^-.

Example 2. In order to show how diagnoses are computed, we consider the example depicted in Fig. 1 with an observation given by the marking μ^{OBS}, where $\mu^{OBS}(m_1) = c$ and $\mu^{OBS}(m_2) = a$ (For simplicity, we use the notation of sets in which the elements are couples of places with its markings, and so: $\mu^{OBS} = \{(m_1, c), (m_2, a)\}$). A classification of those observed manifestations can be that in which $M^+ = \mu^{OBS}$. According to the classification of the observed manifestations, the diagnosis problem definition will be given as it has been discussed in [3], where the authors suggest that for the same observation, we may have a spectrum of definitions varying from a pure consistency-based to a pure abductive diagnosis. Fig. 2 shows the backward reachability graph corresponding to the case that we have where $\mu = \mu^{OBS}$. Arcs of the graph are labelled by steps (the set of fired transitions in backward fashion). The framed transitions represent forced ones. Notice that t_1 is forced when the place c_{12} becomes marked with a color a (that is present in $b\_FW(t_1)$). Moreover, there is a path in the backward reachability graph whose terminal node is an inconsistent marking where the place s_1 has two different markings.

The obtained solutions are $\mu_1 = \{(c_1, a), (c_2, a)\}$ and $\mu_2 = \{(c_1, a), (c_2, a), (c_3, a)\}$. It should be noticed that $\mu_1 \subset \mu_2$, thus, $\mu^{ini} = \{\mu_1\}$ (a solution to a diagnosis problem have to be minimal). For our case, μ_1 is a consistent solution.

6 Related work

During the last decade, several model-based approaches and frameworks have been proposed, as improved ones, to solve fault diagnosis problem. The majority of them perform such a reasoning on the basis of PNs and their different

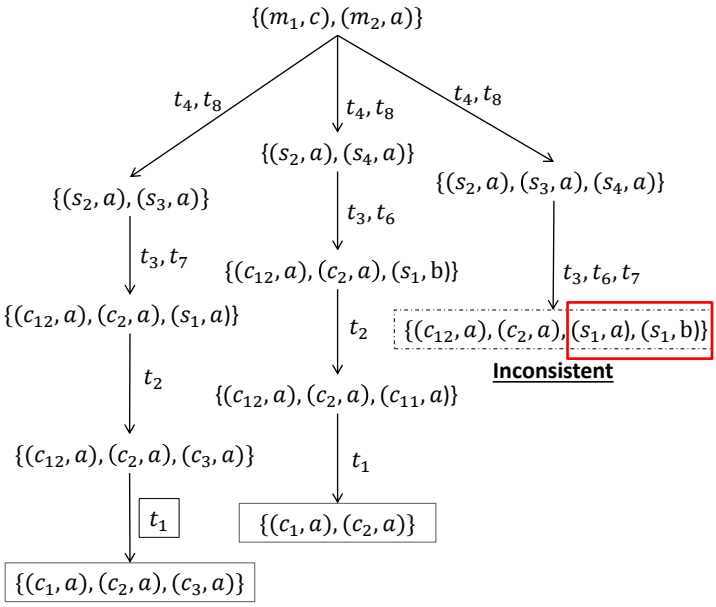


Fig. 2. A backward reachability graph.

extensions as suitable formalisms because of their mathematical and graphical representation. Among the recent works, we recall those proposed in the context of discrete event systems. The approach proposed in [6] exploits basis markings for an on-line diagnosis using labelled PNs. The main advantage of such an approach is that the reachability space is more compact. In the field of CPNs, [7] presents a CPN version of the diagnoser introduced in [8]. It should be known that such a diagnoser is constructed on the basis of a labelled PN model of the diagnosed system. The approach is, then, extended to implement a modular diagnoser for distributed and large systems. [9] deals with the problem of diagnosis in workflow processes, where, the author defines a CPN fault model, in which the faults can be of two sources: faulty input places or faulty transition modes. Places represent the system variables that can be of a correct, faulty or unknown status. The diagnosis problem reasoning is accomplished backwardly by solving its corresponding symbolic inequations system. The authors of [5] develop a backward reachability analysis method for CPNs. Such a method performs a structural inversion [3] of the CPN model, and so, analysing the CPN model backwardly becomes a forward analysis of its inverted one.

[3] Such an inversion preserves the original model proprieties.

Our proposal differs from these by focusing on a particular side, when modelling a system, that is the causal behaviour. In this scope, we are interested in the approach presented in [2] for centralised diagnosis performed on the basis of BPNs, and its corresponding distributed one defined in [10]. CCPNs are defined as a particular and simplified class of CPNs for describing the causal behaviour, as well as, simplifying the analysis methods.

7 Conclusion

In this paper, we have presented a new approach based on CCPNs as a particular class of CPNs for representing and diagnosing systems given by means of causal models. In such a net model, we introduced for each of its transitions a matrix describing the functional dependencies among its corresponding places as a way of avoiding the hard problems resulting from the inversion of the arc expressions when analysing the net model backwardly. In order to solve a diagnosis problem given by means of CCPNs, and so, generating the possible diagnoses of a given malfunction, a backward analysis on the reachability graph corresponding to the net model is defined for the case when using matrices. Many issues remain to be investigated. Among those we mention: the use of structural analysis as an alternative of reachability graphs that are characterized by the combinatorial explosion during the consistency checking of the generated initial markings; and extending the approach for the distributed systems where there is a set of subsystems each of which is given by a CCPN having its local observation and so local diagnoses, the main question is that how can interactions be managed to achieve global consistency between the different local diagnoses.

References

1. L. Portinale, "Verification of causal models using petri nets," *International Journal of Intelligent Systems*, vol. 7, no. 8, pp. 715–742, 1992.
2. ——, "Behavioral petri nets: a model for diagnostic knowledge representation and reasoning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 27, no. 2, pp. 184–195, 1997.
3. L. Console and P. Torasso, "A spectrum of logical definitions of model-based diagnosis," *Computational intelligence*, vol. 7, no. 3, pp. 133–141, 1991.
4. K. Jensen, "Coloured petri nets and the invariant-method," *Theoretical computer science*, vol. 14, no. 3, pp. 317–336, 1981.
5. M. Bouali, P. Barger, and W. Schon, "Colored petri net inversion for backward reachability analysis," *IFAC Proceedings Volumes*, vol. 42, no. 5, pp. 227–232, 2009.
6. M. P. Cabasino, A. Giua, M. Pocci, and C. Seatzu, "Discrete event diagnosis using labeled petri nets. an application to manufacturing systems," *Control Engineering Practice*, vol. 19, no. 9, pp. 989–1001, 2011.
7. Y. Pencolé, R. Pichard, and P. Fernbach, "Modular fault diagnosis in discrete-event systems with a cpn diagnoser," *IFAC-PapersOnLine*, vol. 48, no. 21, pp. 470–475, 2015.

8. M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Diagnosability of discrete-event systems," *IEEE Transactions on automatic control*, vol. 40, no. 9, pp. 1555–1575, 1995.
9. Y. Li, "Diagnosis of large software systems based on colored petri nets," Ph.D. dissertation, Université Paris Sud-Paris XI, 2010.
10. H. Bennoui, "Interacting behavioral petri nets analysis for distributed causal model-based diagnosis," *Autonomous agents and multi-agent systems*, vol. 28, no. 2, pp. 155–181, 2014.

Simulating Multiple Formalisms Concurrently Based on Reference Nets

Pascale Möller, Michael Haustermann, David Mosteller, and Dennis Schmitz

University of Hamburg, Faculty of Mathematics, Informatics and Natural Sciences, Department of Informatics, http://www.informatik.uni-hamburg.de/TGI/

Abstract Modeling complex systems nowadays requires a combination of techniques to facilitate multiple perspectives and adequate modeling. Therefore, UML and other formalisms are used to enrich the portfolio of Petri nets. Often the different models are transformed into a single formalism to simulate the resulting models within a homogeneous execution environment. For UML, the mapping is usually done via the transformation to some programming language. Anyhow, the problem with generative techniques is that the different perspectives that are provided by the applied modeling techniques can hardly be retained once the models are transformed into a single formalism.
In this contribution we elaborate on how multiple formalisms can be used together in their original representation. One of the main challenges for our approach is the provision of means for coupling mentioned formalisms so they can be executed together. We utilize the synchronization features of Reference Nets to couple multiple modeling techniques. This results in a simultaneous and concurrent execution of models featuring a combination of multiple modeling formalisms. A finite automata (FA) modeling and simulation tool is presented to showcase the principle concepts and options that are gained by our results.

Keywords: Petri Nets, Multi-Formalism, Model Synchronization, Reference Nets, Finite Automata

1 Introduction

Modeling complex systems requires a separation of concerns and demands support for taking multiple perspectives on the system including various levels of abstraction. This is achieved by combining several modeling techniques.

Looking back at the original ideas of Carl Adam Petri [27], automata are enhanced by a communication mechanism, thus introducing the modeling of concurrency. The new formalism, he calls "net", facilitates the modeling of additional concepts: locality (of time and place), asynchronism and concurrency. With his conceptual extension to the formalism comes a shift of perspective. In consequence, nets may serve for other purposes than automata.

Petri nets in their various forms have proven to be an adequate technique to model, analyze and understand concurrent systems. They provide an operational

semantics, and by utilizing high-level Petri nets, modelers have a technique at hand to cover multiple abstraction levels and perspectives. Anyhow, Petri nets are not always the optimal solution for every modeling task. We agree with Milner, who says: "I reject the idea that there can be a unique conceptual model, or one preferred formalism, for all aspects of something as large as concurrent systems modeling" [25, p. 78].

Depending on the context and the application area, various requirements to the modeling technique have to be met. Extending a formalism (or creating a new one) allows to shift the focus to other aspects, thus providing a different abstraction. Multiple techniques may be combined to facilitate various perspectives in order to cover different levels of abstraction (vertical) or for a combination of complementing views (horizontal).

Today, modeling techniques are quite elaborate, and the means for modeling are systematically developed and researched. The purposes for constructing *conceptual models* and their usage areas are manyfold. Krogstie [19, p. 3] has put them into categories: models may be used for system (forward) design, for the (computer-assisted) analysis, to support communication, for quality assurance, model deployment and activation, or just for making sense out of something.

The complexity of systems has dramatically increased since the beginning of computer science. This raises the necessity to provide several perspectives at the same time. Of course, taking different perspectives makes no sense at all if they can not be somehow related, which means they have to be linked together.

Standard modeling languages like UML (Unified Modeling Language) offer a portfolio of techniques to provide multiple modeling perspectives covering various levels of abstraction. Model driven development approaches facilitate the development of domain specific languages that can be related on the basis of meta-models. Usually, they are generated to a single target language, which comprises a loss of the desired abstraction level and an alteration of the perspective. Model simulation environments support the execution of models, but models usually do not provide the means to be used in combination.

The concurrent execution of multiple modeling techniques in their original representation, using explicit model coupling, are promising in order to combine the advantages of linking models and direct simulation. Consequently, tool support is required not only to support the modeling of multiple perspectives, but also to integrate the different views and to keep them consistent with each other.

We identify the following two main challenges. (1) For the generic coupling of multiple modeling techniques an adequate mechanism has to be found. (2) How is it possible to simulate coupled models from various modeling techniques in one environment concurrently without losing the original representation?

In this work we propose a conceptual extension of current modeling techniques by synchronous channels [7] as they are provided in the context of Reference Nets. Synchronous channels can supply synchronous communication between models in order to exchange data or control other models. Furthermore, we propose to map the constructs of the modeling techniques to Reference Net constructs. It should be noted that this mapping is not equivalent to the trans-

formation that we reject above, since the original representation of the model is preserved and even visually visitable during simulation. For modelers it should appear as if a model is being simulated in its original modeling technique. In order to achieve this, we propagate the simulation events of the underlying Reference Net back to the original model.

2 Proposal for Coupling through Synchronization

In this section we briefly introduce the conceptual and technical background of our work and present a simple introductory example of the coupling of multiple formalisms. First, we present the Renew environment for modeling and execution of Reference Nets and other modeling techniques. (Java) Reference Nets as our main modeling and simulation formalism are introduced afterwards. Last, with a simple example, we demonstrate our idea of multi-formalism execution, which is presented in general in Section 3.

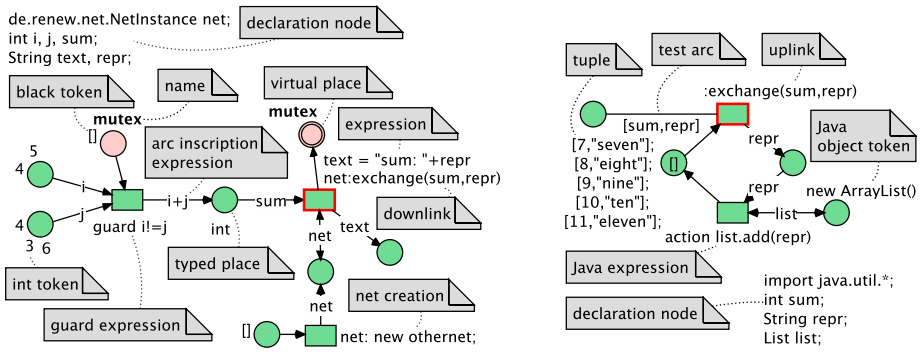
2.1 Renew

Renew is a continuously developed extensible modeling and simulation environment for Petri nets and other modeling techniques [21]. Due to its recent enhancements and extensions it has evolved into an IDE (integrated development environment) for the development of Petri net-based software [5]. Renew is focused on Reference Nets and has full support for that formalism in terms of modeling and execution with or without graphical feedback. Benefiting from its plugin architecture, over the past years, Renew has been extended with multiple modeling techniques and formalisms. With recent efforts regarding model-driven language engineering [26], Renew has become an environment for the development of modeling techniques and tools as well. The current development version has full support for the concurrent and coupled simulation of multiple formalisms. Renew is written in Java and available for various platforms such as Linux, Mac OS and Windows including the source code.[1]

2.2 Reference Nets

The (Java) Reference Nets formalism [20] is a high-level Petri net formalism that combines the nets-within-nets paradigm [29] with synchronous channels [7] and a Java inscription language. This formalism makes it possible to build complex systems using dynamic net hierarchies. The nets-within-nets concept is implemented using a reference semantics so that tokens can be references to nets. With the Java inscription language, it is possible to use Java objects as tokens and execute Java code during the firing of transitions. The synchronous channels

[1] The full multi-formalism feature will be part of Renew version 2.6 and then be available at `http://renew.de`. The current development version can be found at `http://paose.net/wiki/MultiFormalism`.



(a) somenet (b) othernet

Figure 1: Example Reference Net system showcasing selected features (image and description adapted from [6], image originally from [12, p. 77])

enable the synchronous firing of multiple transitions distributed among multiple nets and a bidirectional exchange of information. An introduction to Reference Nets is available in the Renew manual [22], the formal definition can be found in [20] (in German). In the following we give a brief overview over the features of Reference Nets based on an example.

The Reference Net system depicted in Figure 1 exhibits a large part of the constructs of Reference Nets and features those that are relevant for this contribution. It consists of two corresponding nets, *somenet* and *othernet*. The places can hold black tokens, elementary data types, or references to complex data types or embedded Reference Nets. Java types are imported and declared in the declaration node. Net instances are created from net templates using the `new` keyword as shown in the lower part of Figure 1a. Transitions have a flexible inscription language featuring guards, synchronous channels, and Java expressions. Places and arcs can also hold inscriptions. Virtual places are virtual copies of places, where the same semantic place may have multiple graphical figures in order to prevent long and crossing arcs.

The two highlighted transitions from *somenet* and *othernet* form a synchronous channel, which always consists of two parts: *downlink* and *uplink*. The downlink must hold a reference to the net containing the uplink. The net *somenet* in Figure 1a holds the `:exchange(sum,repr)` channel's downlink and a reference (`net`) to the net *othernet* in Figure 1b, which contains the uplink. Even though the invocation of the synchronous channel is directed, due to the required reference, the information exchange is bidirectional.

The unification algorithm searches for bindings of a transition that satisfy the possible assignments of variables with respect to the declared inscriptions on transitions, arcs, and places. If both transitions participating in a synchronous channel are enabled, they can fire synchronously and exchange information in both directions. In this example the `:exchange(sum,repr)` channel unifies the

sum of the values of `i` and `j` bound to the variable `sum` in *somenet* with the variable `sum` from the tuple of `sum` and `repr` from *othernet*.

The mechanism of synchronous channels provides powerful features for the coupling of multiple models. As described above, a synchronous channel consists of a pair of up- and downlink. The main reason for this is an efficient implementation (with mostly polynomial complexity instead of exponential complexity) that has been described in [20]. If multiple uplinks of one net can participate in the firing of synchronized transitions, one of the possible uplinks is chosen non-deterministically. For each downlink (and its parameters) there is exactly one uplink that will be bound when a transition is fired, and both must participate since the firing of the transitions is atomic. In the following we will briefly sketch our approach to coupling multiple modeling techniques by introducing our running example, before we generalize from this idea in order to develop our conceptual approach.

2.3 Coupling Finite Automata with Reference Nets

The coupling of finite automata and Reference Nets serves as a simple example for coupling multiple formalisms. We facilitate the coupling through enhancing the finite automata formalism with synchronous channels. More precisely, we develop a concept to synchronize finite automata state transitions with the firing of Reference Net transitions.

In this section we outline the coarse idea by presenting a useful example for the coupling of finite automata and Reference Nets. The general concept for the coupling and simulation of multiple formalisms is described in Section 3. An exemplary implementation that allows the simultaneous and synchronized execution of both – finite automata and Reference Nets – is described in Section 4.

One application area of multi-formalism simulation is the controlling of systems to avoid unwanted behavior such as deadlocks or security violations. To enforce orderly behavior, it is possible to use a controlling instance to restrict the possibilities of the controlled system. Ezpeleta et al. use controllers with Reference Nets in such a way [10]. Finite automata are well-suited for this purpose because they provide an intuitive description of the desired behavior.

We present a simple example to motivate a goal of controlling nets – avoidance of deadlocks and unwanted situations, as mentioned by Burkhard [4]. The Reference Net in Figure 2 depicts a simple production and consumption process. The consumption (lower right part) requires at least one preceding production (upper right part) to prevent the system from running into a deadlock.

At the left hand side there are three manually fireable transitions that each instantiate one of the controller automata shown in Figure 3. A reference token can be removed anytime by the centered transition. Producing puts a token into the storage place. Consuming is processed in two steps. In the first step the consumption process is entered. In a second step a token from the storage is consumed. If there is none, the net is stuck in a deadlock.

To avoid a deadlock, one may use a finite automaton. Three deadlock-avoidance strategies are shown in Figure 3. Strategy *avoid* (Figure 3a) constrains the

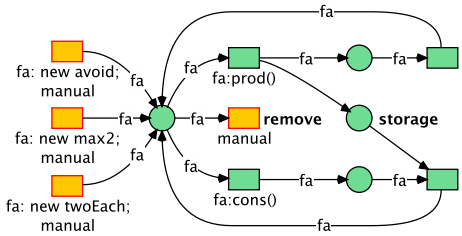
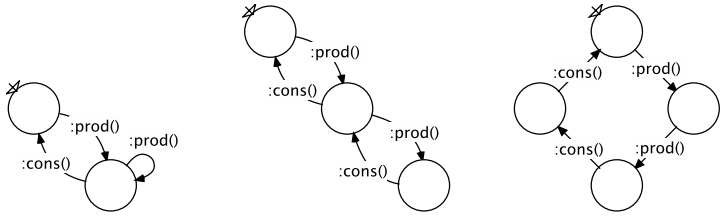


Figure 2: Reference Net with potential deadlock

net to produce at least once before each consumption. Strategy *max2* (Figure 3b) limits the number of tokens in the storage to 2, so that a consumption process takes place at least after every second production. The third strategy *twoEach* (Figure 3c) predefines the order to two productions followed by two consumptions repeatedly. With all three strategies a deadlock is avoided.



(a) avoid (b) max2 (c) twoEach

Figure 3: Different deadlock avoidance strategies

3 Concept for Multi-Formalism Simulation

In the following we elaborate on our approach to multi-formalism simulation that we sketch in Section 2. We will generalize the idea of using finite automata to control and visualize parts of a system. We present a method to link multiple formalisms through synchronized actions on the basis of Reference Nets.

3.1 Coupling via Synchronous Channels

In order to capture the overall structure and behavior of complex software systems, several modeling techniques are applied in combination. Coupling of models supports the modeling process in general by providing modeling techniques that exactly match the requirements of the modeling purpose. Each created model covers a distinct yet partly overlapping perspective of the system. All models need to be integrated in a consistent way to cover the complete system.

In a common setting – e.g. when using UML – the models are, more or less, modeled in isolation. The relations to other modeling techniques and therefore between the created models are not shown explicitly. While at first sight this makes it easier for the modeler, this is a problem. Modelers must know how models interact with each other. Usually, this relation is established by the compiler of the models if the models can be used directly for code generation. In most modeling environments this is not the case. In consequence, the models are complemented with implementation details to facilitate their execution.

As motivated in the introduction, the transformation of models to a single target language involves a loss of perspectives when looking at the executed system. While, of course, a common execution language may work in the background, we propose to directly execute the models and to make the coupling explicit in order to retain the perspectives. The combined simulation in their original representation requires an operational semantics of the applied techniques and a mechanism for the coupling of models.

Coupling multiple techniques requires considerable conceptual and technical support. We use, in addition to the operational semantics of the modeling techniques, an execution environment that properly supports multi-formalism simulation: Renew. A drawback of the explicit coupling of multiple modeling techniques is that it involves an extension of the modeling languages.

Direct simulation is not applicable for every modeling technique. For example the execution of a solely structural diagram seems not to be useful. In this contribution we mainly consider behavioral techniques that are discrete and state-based. This covers many of the UML behavioral diagrams and process modeling languages, such as BPMN and EPC.

3.2 Model Coupling with Graphical Feedback

In our group we have studied the necessary and sufficient solutions to implement modeling techniques within our framework(s). To extend high-level Petri nets by synchronous channels, elaborate algorithms were needed. The specification, design and implementation was a highly complex task. However, on top of this a very powerful semantics for the description of other modeling techniques is available now. With previous contributions we have shown how a new formalism can be developed on the basis of Renew by providing operational semantics through a mapping to Petri nets [16,26]. In this contribution we propose to create formalisms that use a mapping to Reference Nets in the background in order to provide the operational semantics for the modeling technique but present the original representation to the user.

Figure 4 summarizes the general idea of our approach to multi-formalism modeling and execution. The upper part (Graphical Layer) of the figure contains the graphical models (model drawings) and model instance drawings that are visible to the user and permit user interaction. Model drawings are artifacts that are created from a graphical editor within Renew or may be imported from an external tool. The instance drawings reflect the simulation state to the user and allow control over the simulation, e.g. by triggering a certain simulation step.

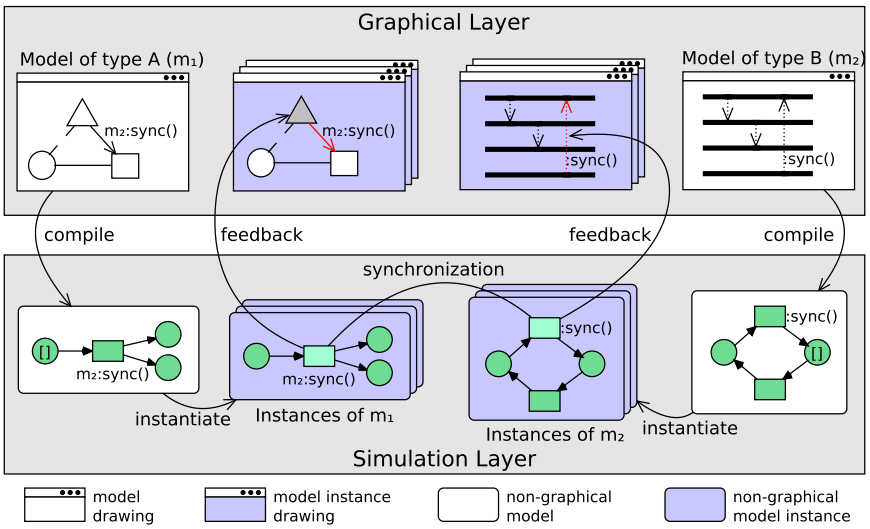


Figure 4: Conceptual model of the model synchronization

The displayed model drawings on the top-left and top-right hand side are arbitrary in a sense that they do not have a real application or semantics and serve as representative for any modeling technique a modeler may want to use. A simple example of how a concrete modeling technique is implemented is provided for communicating automata in Section 4. For a specific modeling technique a mapping from the constructs of the modeling technique to Reference Net constructs is needed in order to obtain an executable model. This may be compared to a code generation approach. The development of such a mapping is a part of the modeling effort, a developer of a modeling language is obliged to perform.

Graphical models are compiled into non-graphical Reference Nets, which can be instantiated and executed in the Renew simulator (Simulation Layer). Synchronization is performed on the level of the Reference Net instances (originating from the same or other modeling techniques). A mapping of the dynamic view of generated Reference Nets back to the modeling technique constructs even allows a direct feedback. The presented solution enables the simulator to pass simulation events from the Reference Net to the original model, e.g. to highlight a corresponding graphical figure. Introducing an additional Reference Net layer on top of the simulator has advantages, but also comes with a few restrictions.

3.3 Discussion

The best solution for a modeling and execution tool with respect to performance is an optimized implementation with a generic coupling mechanism. However, such an implementation from scratch is hard and time consuming. Our approach based on Reference Nets simplifies the development of executable, coupled modeling techniques with a powerful and yet easy to use mechanism for synchro-

nization. Due to reasonable hardware and an efficient implementation of the Reference Net formalism, this approach is applicable without a huge drawback in terms of performance, and the Reference Net formalism is an adequate language for the definition of an operational semantics.

A benefit of our concept is that the means for synchronization are made utilizable. This makes it easier to develop formalisms and to concentrate on the modeling language engineering. It supports a prototypical approach because language constructs may be designed in their Reference Net representation to be later on replaced with the constructs of the formalism in development. Synchronous channels provide a common conceptual basis for combining multiple formalisms. All formalisms share the same mechanism for communication and are consequently designed to be used in combination. Users directly profit from the *native* feedback in comparison to generative approaches.

Reference Nets are well-suited to cover multiple properties of modeling techniques, especially those of discrete, state-based behavioral techniques that we consider in this contribution. In order to simulate an arbitrary number of models concurrently, the execution language has to provide an intuitive mechanism to support the implementation of concurrency. The requirement for atomicity of activities and resource allocation emerges from the possibility of concurrent activities in a set of models. The simulated models should not effect each other, unless it is explicitly intended via synchronization. Thus, the execution language has to support strict data encapsulation for models in simulation. For process modeling languages often dynamic hierarchization is required (e.g. for subprocesses in BPMN 2.0). Reference Nets as Petri net formalism provide a powerful mechanism to implement concurrent behavior, resource allocation and atomicity of actions. Due to the nets-within-nets concept, encapsulation of simulated models and dynamic hierarchization is provided naturally by Reference Nets.

The utilization of synchronous channels as a coupling mechanism implies that the coupling always has to be synchronous, but the implementation of asynchronous coupling is possible as well. Two channels with an intermediate buffer-place result in asynchronous behavior.

Another property that is inherent in many modeling techniques is time. An extension of Reference Nets with timed expressions exists [22]. The presented concept is thus applicable for timed modeling techniques using this formalism as basis. Renew supports the execution of these nets. However, the formalism can only be simulated sequentially.

Generating models into a single formalism also has the advantage of being able to verify within a single formalism. For the intermediate format of Reference Nets, however, there are not many verification tools available yet. In general the usage of a single formalism seems to be valuable. With some of our tools, that are currently under development, we can generate a reachability graph for some restricted net formalisms.

The use of Reference Nets as coupling mechanism comes with some limitations. As described in Section 2.2, one synchronous channel, comprising up- and downlink, synchronizes exactly two transitions. There exist no means for global

synchronization (due to the basic idea of the locality principle of a transition) and consequently, there is no broadcast communication synchronizing all transitions by default. A synchronous channel only provides the synchronized action of two elements. The synchronization of more than two transitions is accomplished by combining multiple channel inscriptions on one transition (having at most one uplink, but several downlinks). In many cases, these restrictions do not constrain the general possibilities of modeling or easy alternatives exist: e.g. when a model instance can/shall not hold references to other models, communication may be provided by a system net that holds references to all participants.

The limited scope of variables to one transition and the connected arcs is a sensible property for Petri net formalisms, because it fits the general concept of locality, which is inherent in Petri nets. For other modeling techniques this may be a limitation. In many cases, Reference Nets are still adequate to provide operational semantics to modeling techniques with global variables. The use of a single place for each variable and an arc connection to these places for every transition can be used to simulate global variables. However, the variables in Reference Nets are immutable in a sense that there is unification but no assignment of variables. Therefore, the use of additional variables is required sometimes. This said, we move on to the development of a formalism that profits from the introduced concept for multi-formalism simulation.

4 Development of a Finite Automata Plugin

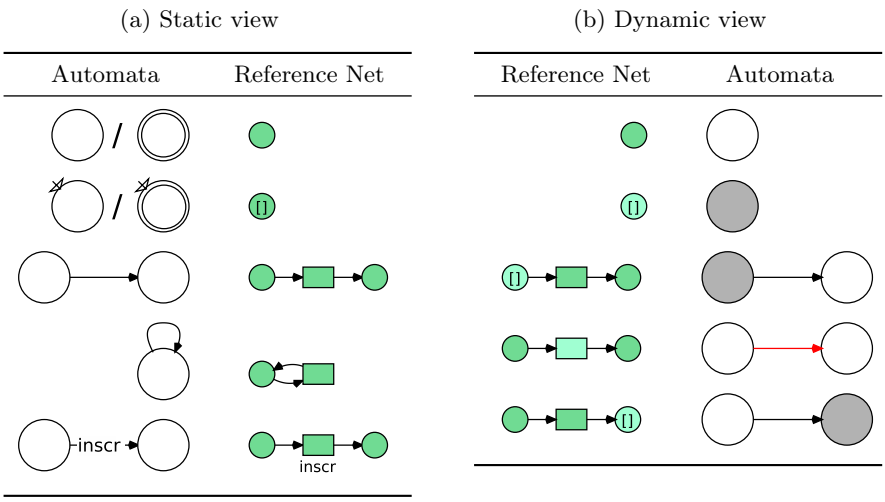
In order to demonstrate the principle concepts that are applied to couple models we describe the development of the Finite Automata plugin (FA plugin). This plugin extends Renew with the capabilities for modeling and simulating finite automata that have the ability to synchronize with Reference Nets.

At first sight it may seem odd to use finite automata next to Petri nets because they actually provide a similar perspective. Simple Petri nets already, and Reference Nets a fortiori, are more expressive than finite automata. Still, finite automata offer an intuitive reflection of a systems state, which is especially helpful when multiple levels of abstraction of a complex system shall be covered.

4.1 Requirements

We derive the following requirements from our goal of extending Renew with support for modeling finite automata and for simulation on the basis of Reference Nets. A concept of finite automata for simulation and synchronization with Reference Nets is necessary. For this purpose, the finite automata modeling language is to be extended. Also, during execution, an automaton's state shall be *visually* inspectable. Furthermore, the editor has to provide usability features to support efficient modeling.

Table 1: Mapping of finite automata and Reference Net constructs



4.2 Concept and Design

The modeling of finite automata is provided by the FA plugin, which was available prior to this work. In its previous version it was restricted to offer some capabilities for drawing finite automata, without the possibility of simulation. As Renew is natively capable of simulating Petri and Reference Nets, we extend the existing FA plugin with simulation and synchronization capabilities.

Reference Net Based Semantics As already mentioned in Section 3, we propose to provide semantics via a mapping to Reference Nets in order to exploit their synchronization features. Additionally, we have to manually implement the visual representation of the simulation state so that the user has feedback in the original representation of the modeling technique (i.e. the state of the simulation of the Reference Net at runtime has to be mapped back on the automaton).

The mapping of finite automata to Petri nets is straightforward as displayed in Table 1a. A finite automaton's state is mapped to a Reference Net's place. We do not distinguish between regular and end states, because we are more interested in the dynamic behavior than in the investigation of the formal language an automaton generates. Start states are mapped to places that are initially marked with a black token (represented as `[]`). A state transition between two states is mapped to a transition that is connected to the places representing the corresponding states. This also holds for the special case of a self-loop. The inscriptions of the state transitions are mapped to inscriptions of net transitions.

In addition to the static mapping, we need to provide a visualization of the current simulation state by mapping the dynamics of the Reference Net back to the automaton. This mapping is depicted in Table 1b. An empty place is

mapped to a non-activated state. A place that contains a token is mapped to an activated state (i.e. the current state in a DFA), which we highlight by a gray filling. The firing of a transition (represented by a highlighted transition) is mapped to a red highlighted state transition.

Synchronization of Finite Automata and Reference Nets We extend finite automata by synchronous channels to couple them with Reference Nets (see Section 2.3). The direct mapping of inscriptions from state transitions to inscriptions on Reference Net transitions makes the use of synchronous channels and other Reference Net inscription types possible. This is due to the compiler, which is handling the inscriptions as if they were attached to a Reference Net. Consequently, Reference Nets or other modeling techniques that provide compatible synchronous channels can synchronize with finite automata in the same way.

4.3 Modeling Capabilities

In this section we focus on the support for modeling finite automata with the FA plugin. This comprises the drawing capabilities of the editor and the means for the annotation of FA constructs with Java inscriptions.

Basic Modeling The FA plugin can be used to model finite automata using the constructs that are depicted in Table 1a: start states, end states, start-end states, neutral states and state transitions (between states). Because the FA plugin is implemented as a Renew plugin it inherits modeling capabilities and usability features. These features are complemented with some useful improvements such as arc drawing handles or interchangeable state representations. The FA plugin supports modeling of nondeterministic finite automata as well as deterministic finite automata. We limit the quantity of start states to one, so that the semantics remains simple. Drawings may be ex- and imported to the JFLAP[2] format and to a textual representation in addition to the standard formats of Renew.

Reference Net Inscriptions The inscriptions of our finite automata are inherited from Reference Nets. Consequently, the syntax and the semantics are similar. The inscription types, that are relevant for the simulation of finite automata along with Reference Nets, are depicted in Figure 5. In general, more inscription types are available. These are described in detail in [22, p. 48 ff.].

Nevertheless, modelers of our extended finite automata have to consider some differences to the modeling of Reference Nets. Reference Net inscriptions can only be – in a sensible way – attached to state transitions. This means, Reference Net inscriptions that are attached to states are not handled by the compiler and thus these are *not* executed by the Renew simulator in contrast to Reference Nets where Java inscriptions to places are possible. Another important consideration is that variables in finite automata can not be stored and only accessed arc-locally (see Section 3.3). We prototypically implemented the feature to provide

[2] JFLAP [18] is a modeling tool that is mainly concerned with teaching basics of theoretical computer science.

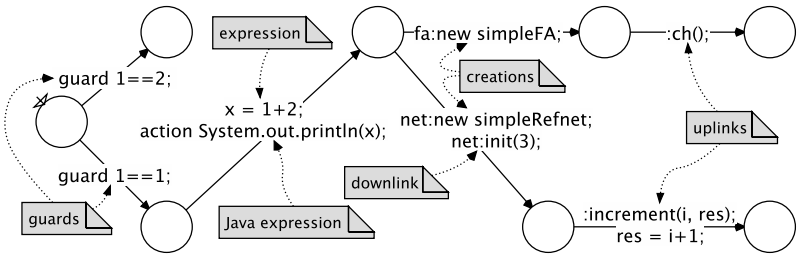


Figure 5: Available inscription types for finite automata

modelers with the capability to initialize the finite automata with diagram-global variables already. So far, this implementation is experimental.

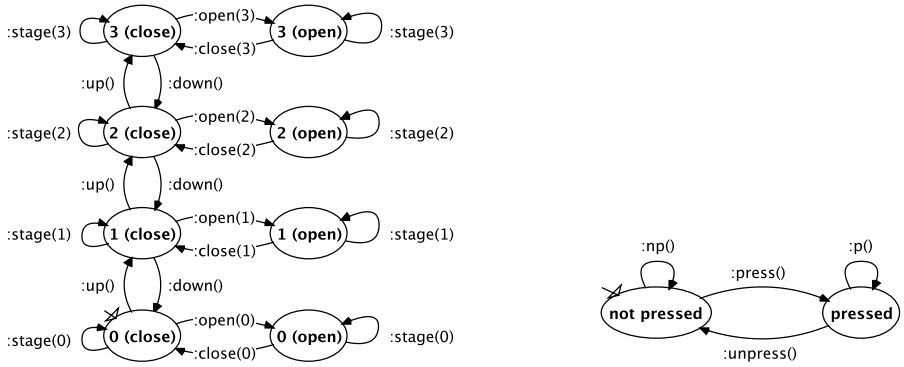
We forgo the description of each inscription that is illustrated in Figure 5 because most of them are inherited from Reference Nets. The synchronous channel inscriptions are of specific relevance in this context. These can be used analogously to the Reference Nets (see Section 2.2). Besides of calling uplinks of other models, our extended finite automata are also capable of providing uplinks.

4.4 Formalism Implementation

The implementation of the formalism touches multiple layers of Renew: (1) graphical layer, (2) shadow layer and (3) compilation layer. We identify the following four main tasks: (a) implement a compiler that translates the graphical representation into a non-graphical data structure (called *shadow net*), (b) develop a mapping from finite automata concepts to Reference Net concepts, (c) implement a second compiler that compiles the finite automata constructs according to the previously developed mapping to Reference Net constructs (called *compiled nets*) and (d) implement the return of feedback from simulation events to a graphical representation. With the completion of these four tasks, the FA plugin for Renew provides the concurrent simulation and synchronization of finite automata and Reference Nets. The FA plugin as described in this section is part of the Renew package referenced in Section 2.1.

5 Lift Application

In this section we present a slightly larger example for the use of multi-formalism execution. The presented system is a model of a lift, which is partly created as automaton and partly as Reference Net. The lift can move up- and downwards between four floors and open its door on each of the floors. It is possible to call the lift on a floor by pressing a button on the respective floor. For reasons of simplicity, we assume that pressing the button on one floor has the same effect as pressing the button for the destination floor from inside the lift. Thus, we do not model the touch panel inside the lift explicitly. In this scenario the status of



(a) Lift automaton (b) Floor status automaton

Figure 6: Automata models of the lift example

the lift and the status of the button on each floor is simple. These are systems without any concurrency and with a defined state. Therefore, these components are modeled as finite automata. The complex part in this scenario is the control mechanism that ensures a reasonable serving strategy. Consequently, it is realized as a Reference Net. Thereby, we use adequate modeling techniques for all parts. The control mechanism as independent component can be exchanged by another version that uses a different serving strategy.

Automata Models The automata models as depicted in Figure 6 have multiple state transitions attached with uplink inscriptions in order to be synchronized with the Reference Net. With the loop at each state, it is possible to implement state dependent behavior (e.g. opening the door is only possible when the button on the respective floor is pressed).

Reference Net Control The Reference Net in Figure 7 implements a lift control that serves a floor when the lift is requested by a pressed button and prefers to keep the movement direction. On the top left hand side of the net, instances of the automaton models are created (one instance of lift, four instances of floor). On the top right hand side there are four transitions that trigger a push of the button in one of the floors. For technical reasons it is not possible to fire state transitions in automaton models, yet. Hence, these transitions in the control net are a simple solution to overcome this technical restriction.

The actual lift control is divided vertically into the four floors. Each floor consists of three to four control places (green places) representing the lift on the respective floor on its way downwards (leftmost place), on its way upwards (rightmost place) or with an open door (places in the center). Changes to the lift and floor models or state queries to these models require access to the lift and floors places. This is achieved by using virtual places.

The lift is initialized with closed doors in the ground floor. With Transition *o* in the bottom, the door is opened initially. Transition *cu0* closes the door of the

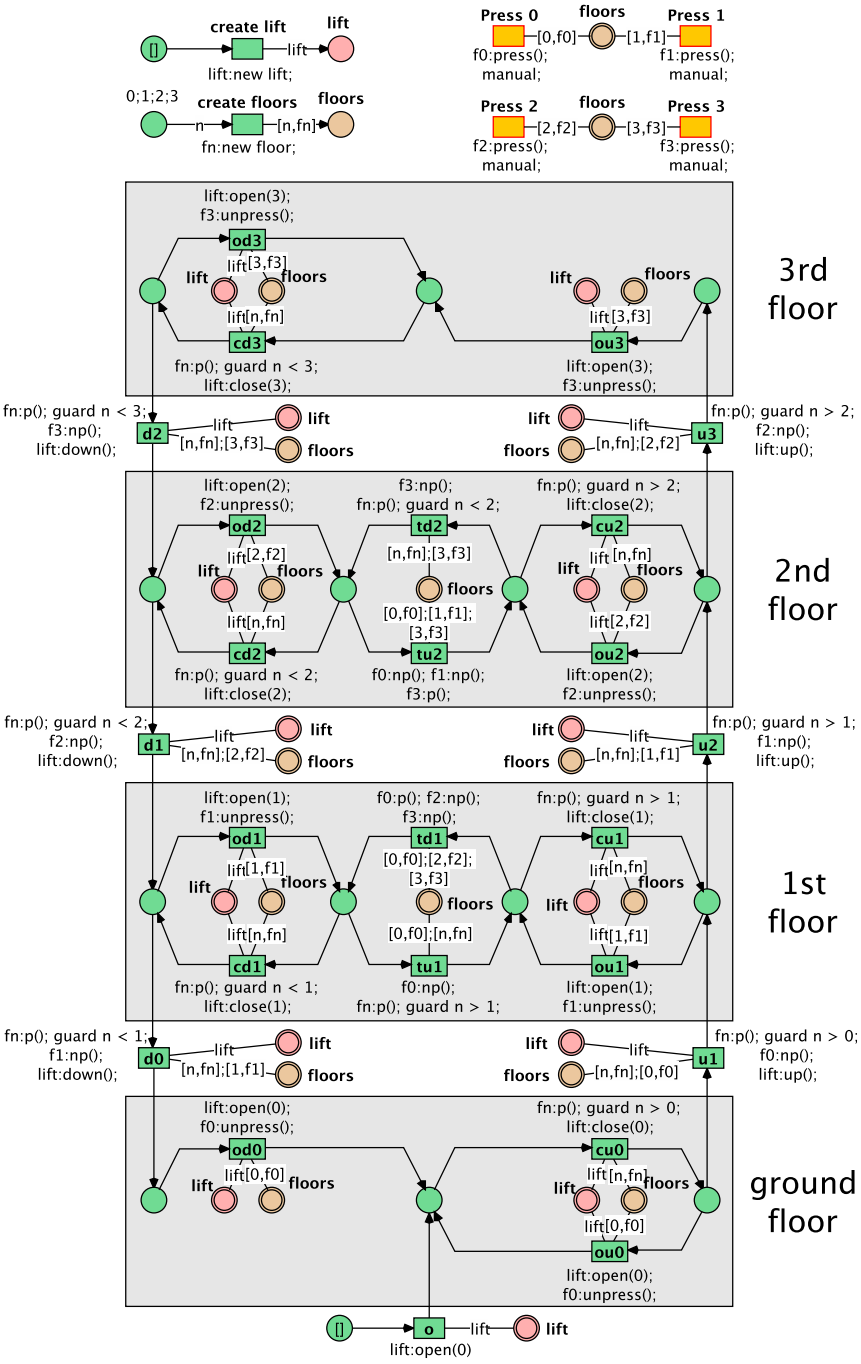


Figure 7: Reference Net for controlling the lift

lift (`lift:close()`), which can then start its way upwards. This is only possible, when the lift was requested on one of the higher floors (i.e. the button is pressed on one of the higher floors, `fn:p(); guard n > 0`). If the door is closed the lift may move upwards one floor with Transition *u1* (`lift:up()`) if the lift was requested from a higher floor (`fn:p(); guard n > 0`) and it was not requested on the same floor (e.g. somebody wants to hop on, `f0:np()`).

With this mechanism, the lift can move upwards until it reaches a floor where the button is pressed (e.g. on the second floor). There, it has to open the door (`lift:open(2)`) and the request for the floor is reset (`f2:unpress()`).

On the first and second floor the lift has two places representing an open door to distinguish the two directions. The lift is allowed to change its direction (with Transitions *td1*, *tu1*, *td2*, *tu2*) if there is no request in the current direction (`f3:np()`) and a request in the other direction (`fn:p(); guard n < 2`).

Analogously to the right hand side of the lift control, the left side implements the controlling of the lift moving downwards.

6 Related Work

In this section we briefly relate our proposal to work in the area of multi-formalism modeling and execution.

Zeigler [30] proposes the multifacetted modeling methodology, which is an approach to simulation modeling by integrating multiple models. The hereby used *Discrete Event System Specification* (DEVS) formalism is capable of constructing coupled models that are composed of atomic DEVS models. In his work an abstract simulation concept for the DEVS formalism was developed. The concept claims to couple serveral simulators for each component in a system of systems to facilitate simulation using a global coordinator for synchronization. Unlike our approach, Zeigler's demonstrations are rather abstract. Our approach is also different from his as we attach importance to concurrent simulation, in particular to true concurrency of Renew. We propose to work with Reference Nets instead of DEVS.

Lara et al. [23] introduce a tool that supports the combined use of multi-formalism modeling and meta-modeling, called AToM^3. Due to the definition of *graph grammar models*, formalisms can be transformed into an appropriate formalism for which simulation is already supported. They suggest the DEVS formalism as the central modeling formalism that can be universally used for simulation purposes. AToM^3 also supports code generation, a meta-modeling layer that can be used to model formalisms in a graphical manner, and the possibility to transform models by preserving the behavior [1]. In contrast to our approach, Lara et al. focus on providing meta-modeling and model-transformation features. The transformed models have to be simulated in an external environment.

Möbius is a tool for modeling and simulating systems composed of multiple formalisms. The project focuses on extensibility by new formalisms and solvers, which is demonstrated in [8]. An *abstract functional interface* is implemented,

which transforms models to framework components to allow for addition of formalisms and interaction between models. Their approach differs from ours in the way of having an overall system state. The Möbius tool enables selective sharing of model states, so that solvers (i.e. simulators) are able to access them.

One practical implementation of a multi-formalism modeling and simulation concept was done by The GEMOC Initiative [13]. This initiative's vision is to advance the research on the coordinated use of modeling languages. They recognized a problem in the unavailability of a generic runtime service for multiple modeling languages. GEMOC Studio is proposed as a tool to create meta-models for both the representation and the operational semantics of modeling languages. Created models of multiple languages can then be executed in coordination, while being debugged and graphically visualized. They use the *Behavioral Coordination Operator Language* (BCOoL [24]) to explicitly specify the coordination patterns between heterogeneous languages. The used execution engine operates as a coordinator of multiple language-specific engines.

Frank [11] proposes a method for multi-perspective modeling that extends the classical approach (of e.g. UML) to conceptual modeling by including the organizational environment. He is also interested in tool support and states the following requirement we share: "A tool environment for enterprise modeling should allow for creating multi-language diagrams, i.e., diagrams that integrate diagrams of models that were created with different DSML" [11, S. 946]. The linking of techniques he proposes is established on the level of a meta-model, but the method lacks an environment to properly support the execution. "However, there are other paradigms that come with specific advantages, too. [...] Languages used for creating simulation models would allow for supplementing enterprise models with simulation features. Petri Nets provide mature support for process analysis and automation" [11, S. 960]. We find the combination of such approaches to modeling language engineering with the provision of operational semantics for a dedicated execution environment most promising.

Jeusfeld [17] proposes the linking of multiple perspectives through declarative constraints in the context of meta-modeling domain-specific languages for the ADOxx platform. He distinguishes the relation between model and external environment from the internal model validity and focusses on the latter. The constraint language is used to define a Petri net firing rule and to sketch a firing rule of BPMN constructs. However, he does not show the combined execution of these formalisms.

There are other researchers who have tried to combine various formalisms with Petri nets. The set of all firings of a Petri net can be considered to be a language. The research of automata and Petri nets as language descriptions has led to the control of Petri nets by (finite) automata [4]. The results suggest that controlling Petri nets through finite automata can be beneficial. A combination of finite automata and high-level Petri nets can be seen as a vertical composition, as both techniques basically provide the behavioral modeling perspective according to Krogstie [19], just on another level of abstraction.

While first the idea of language intersections were of interest, the idea of a *controller* was used in the context of application modeling [28]. A lift system was modeled, however, just the control was addressed and no other modeling techniques were applied. In other research, flexible manufacturing systems (FMS) were used to demonstrate central aspects of control theory (cf. [14] for discrete event system discussion). Most of the authors' work concentrates on techniques that provide one specific modeling perspective. A production system can be controlled by a simple model to ensure that no deadlocks can occur [9,15].

Overall our approach presented here allows to reduce the cognitive load of modelers. The complexity can be reduced by using an appropriate modeling technique, like finite automata, workflows or other modeling techniques like BPMN, eEPCs, Activity Diagrams, etc. However, each formalism needs to be connected via the synchronous channels to be executed within our environment. Systems complying to our interfaces could also mimic our approach if sufficient support for the execution of modeling techniques is available, for example with CO-OPN/2 [2] or the Zero-safe net formalism [3]. Our illustrations of the principle usage in Sections 4 and 5 can be seen as proof of concepts.

7 Conclusion

In this contribution we conceptually present how various formalisms can be used together not only for modeling, but also for simulation while preserving their original representation.

In our opinion, the benefit of using multiple formalisms for the modeling of complex systems can be increased by providing a solution for the simulation of these formalisms. That does not mean that we just intend to transform the models of multiple formalisms into one single formalism, but rather provide simulation feedback for the original representation of the models. Therefore, the various formalisms are mapped to Reference Nets and the simulation events are returned to the models original representation (see challenge (2) in Section 1).

We set up the thesis that the synchronization and data exchange between models of various formalisms should not be achieved through the development of several individual coupling mechanisms. Instead, we represent the opinion, that this should be achieved through *one* coupling mechanism.

We propose the generic coupling of models using the synchronous channels on the basis of Reference Nets (see challenge (1) in Section 1). Regardless of whether the models are of various or the same formalism. Synchronous channels allow for the synchronization of several models, and data exchange in all directions.

As a proof-of-concept we practically demonstrate how this approach can be implemented for the coupling of finite automata and Reference Nets.

In the close future we plan to improve the various concepts and implementations of our approach to support modeling techniques with variables for formalisms that are not capable of storing references by themselves. Furthermore, we investigate how to support the development of techniques through meta-modeling and to provide a close adaption to the simulation environment of Re-

NEW, e.g. to model the techniques themselves, the drawing tools and their operational semantics [26]. In the context of the above mentioned investigation, among others we will develop a RENEW plugin that provides the modeling and simulation of statecharts.

References

1. AToM^3 website. `http://atom3.cs.mcgill.ca`, accessed: 2017-01-19
2. Biberstein, O., Buchs, D., Guelfi, N.: Object-oriented nets with algebraic specifications: The CO-OPN/2 formalism. In: Agha, G., de Cindio, F., Rozenberg, G. (eds.) Concurrent Object-Oriented Programming and Petri Nets, Advances in Petri Nets. Lecture Notes in Computer Science, vol. 2001, pp. 73–130. Springer (2001)
3. Bruni, R., Montanari, U.: Zero-safe nets: The individual token approach. In: Parisi-Presicce, F. (ed.) Recent Trends in Algebraic Development Techniques, 12th International Workshop, WADT'97, Tarquinia, Italy, June 1997, Selected Papers. Lecture Notes in Computer Science, vol. 1376, pp. 122–140. Springer (1997)
4. Burkhard, H.D.: Control of Petri nets by finite automata. Annales Societatis Mathematicae Polonae Series IV: Fundamenta Informaticae VI.2, 185–215 (1983)
5. Cabac, L., Haustermann, M., Mosteller, D.: Renew 2.5 - towards a comprehensive integrated development environment for petri net-based applications. In: Kordon, F., Moldt, D. (eds.) Application and Theory of Petri Nets and Concurrency - 37th International Conference, PETRI NETS 2016, Toruń, Poland, June 19-24, 2016. Proceedings. Lecture Notes in Computer Science, vol. 9698, pp. 101–112. Springer-Verlag (2016)
6. Cabac, L., Haustermann, M., Mosteller, D.: Software development with Petri nets and agents: Approach, frameworks and tool set. Science of Computer Programming (2017), under review
7. Christensen, S., Hansen, N.: Coloured Petri nets extended with channels for synchronous communication. In: Robert, V. (ed.) ATPN. Lecture Notes in Computer Science, vol. 815, pp. 159–178. Springer-Verlag (1994)
8. Courtney, T., Gaonkar, S., Keefe, K., Rozier, E.W.D., Sanders, W.H.: Möbius 2.3: An extensible tool for dependability, security, and performance evaluation of large and complex system models. In: 2009 IEEE/IFIP International Conference on Dependable Systems Networks. pp. 353–358 (June 2009)
9. Ezpeleta, J., Colom, J.M., Martínez, J.: A Petri net based deadlock prevention policy for flexible manufacturing systems. IEEE Trans. Robotics and Automation 11(2), 173–184 (1995)
10. Ezpeleta, J., Moldt, D.: A proposal for flexible testing of deadlock control strategies in resource allocation systems. In: Pahlavani, Z. (ed.) Proceedings of International Conference on Computational Intelligence for Modelling Control and Automation, in Vienna, Austria, 12–14 February (2003)
11. Frank, U.: Multi-perspective enterprise modeling: foundational concepts, prospects and future research challenges. Software & Systems Modeling 13(3), 941–962 (2014)
12. Friedrich, M., Moldt, D.: Introducing refactoring for reference nets. In: Cabac, L., Kristensen, L.M., Rölke, H. (eds.) Petri Nets and Software Engineering. International Workshop, PNSE'16, Toruń, Poland, June 20-21, 2016. Proceedings. CEUR Workshop Proceedings, vol. 1591, pp. 76–92. CEUR-WS.org (2016)
13. GEMOC Initiative website. `http://gemoc.org/index.html`, accessed: 2017-04-02

14. Giua, A., Seatzu, C.: Petri nets for the control of discrete event systems. Software and System Modeling 14(2), 693–701 (2015)
15. Hu, H., Liu, Y., Zhou, M.: Maximally permissive distributed control of large scale automated manufacturing systems modeled with Petri nets. IEEE Trans. Contr. Sys. Techn. 23(5), 2026–2034 (2015)
16. Jacob, T., Kummer, O., Moldt, D., Ultes-Nitsche, U.: Implementation of workflow systems using reference nets – security and operability aspects. In: Jensen, K. (ed.) Fourth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools. University of Aarhus, Department of Computer Science, Ny Munkegade, Bldg. 540, DK-8000 Aarhus C, Denmark (Aug 2002), dAIMI PB: Aarhus, Denmark, August 28–30, number 560
17. Jeusfeld, M.A.: Semcheck: Checking constraints for multi-perspective modeling languages. In: Karagiannis, D., Mayr, H.C., Mylopoulos, J. (eds.) Domain-Specific Conceptual Modeling, Concepts, Methods and Tools, pp. 31–53. Springer (2016)
18. JFLAP website. `http://www.jflap.org`, accessed: 2017-01-18
19. Krogstie, J.: Perspectives to Process Modeling, Studies in Computational Intelligence, vol. 444, pp. 1–39. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
20. Kummer, O.: Referenznetze. Logos Verlag, Berlin (2002)
21. Kummer, O., Wienberg, F., Duvigneau, M., Cabac, L., Haustermann, M., Mosteller, D.: Renew – the Reference Net Workshop (Jun 2016), `http://www.renew.de/`, release 2.5
22. Kummer, O., Wienberg, F., Duvigneau, M., Cabac, L., Haustermann, M., Mosteller, D.: Renew – User Guide (Release 2.5). University of Hamburg, Faculty of Informatics, Theoretical Foundations Group, Hamburg (Jun 2016), `http://www.renew.de/`
23. de Lara, J., Vangheluwe, H.: AToM^3: A tool for multi-formalism and meta-modelling. In: Kutsche, R., Weber, H. (eds.) Fundamental Approaches to Software Engineering, 5th International Conference, FASE 2002, held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8-12, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2306, pp. 174–188. Springer (2002)
24. Larsen, V., Ezequiel, M.: BCOol: the Behavioral Coordination Operator Language. Theses, Université de Nice Sophia Antipolis (Apr 2016)
25. Milner, R.: Elements of interaction - turing award lecture. Commun. ACM 36(1), 78–89 (1993)
26. Mosteller, D., Cabac, L., Haustermann, M.: Integrating Petri net semantics in a model-driven approach: The Renew meta-modeling and transformation framework. T. Petri Nets and Other Models of Concurrency 11, 92–113 (2016)
27. Petri, C.A.: Kommunikation mit Automaten. Dissertation, Schriften des IIM 2, Rheinisch-Westfälisches Institut für Instrumentelle Mathematik an der Universität Bonn, Bonn (1962)
28. Reisig, W.: Embedded system description using Petri nets. In: Kündig, A.T., Bührer, R.E., Dähler, J. (eds.) Embedded Systems: New Approaches to Their Formal Description and Design, An Advances Course, Zürich, Switzerland, March 5-7, 1986. Lecture Notes in Computer Science, vol. 284, pp. 18–62. Springer (1986)
29. Valk, R.: Petri nets as token objects - an introduction to elementary object nets. In: Desel, J., Silva, M. (eds.) 19th International Conference on Application and Theory of Petri nets, Lisbon, Portugal. pp. 1–25. No. 1420 in Lecture Notes in Computer Science, Springer-Verlag, Berlin Heidelberg New York (1998)
30. Zeigler, B.P.: Multifacetted Modelling and Discrete Event Simulation. Academic Press Professional, Inc., San Diego, CA, USA (1984)

Petri Net Inside RFID Database Integrated with RFID Indoor Positioning System for Mobile Robots Position Control

José Jean-Paul Zanlucchi de Sousa Tavares, Rodrigo Hiroshi Murofushi, Lucas Henriques Silva, and Gustavo Rezende Silva

Manufacturing Automated Planning Laboratory (MAPL), Faculdade de Engenharia Mecânica (FEMEC), Universidade Federal de Uberlândia (UFU), Uberlândia, Brazil.
jean.tavares@ufu.br, {hiroshihm, lucashenriquessilva}@gmail.com, gustavorezendesilva@hotmail.com

Abstract. The advent of industry 4.0, internet of things and smart products increase the importance of solution focused on machine-to-machine communication. There is a need for a solution that meets these characteristics and the Petri Net integrated with RFID (PNRD) can reach them. There are a lot of papers connect the Petri Net to RFID by creating the network markings based on the reading of the tags. The PNRD uses the Petri net as the formal data structure to be stored in the tag memory, increasing Petri Net and RFID integration. RFID can also be useful as indoor positioning system or IPS. This work proposes to integrate PNRD and IPS in order to store the object process model in the tag, as well as its position obtained by the IPS can become a prerequisite of the process itself. A case study presents a mobile robot position control based on PNRD and IPS integration.

Keywords: Indoor Positioning System, Radio Frequency Identification, Petri Net Inside RFID database – PNRD, Mobile Robot Position Control

1 Introduction

The advent of industry 4.0 [1], the internet of things (IoT) [2, 3], and smart product [4] make solution focused on device-to-device communication. Continuing to use systems development methodologies focused on human-machine interaction is a challenge in relation to machine-machine solutions. There are highlighted technologies with respect to this machine-machine iteration, especially with regard to operational issues. One of them, pointed as product DNA and information key source, is RFID (Radio Frequency Identification) that allows process and product monitoring, tracking and control [5]. Since the initial RFID application with the Walmart initiative in early 2000's, much attention has been pointed out for this technology. The current RFID market shows that it has been overestimated. Nowadays the RFID implementation is below its potential without process information embedded adding automatic data and process capture.

Numerous researchers that relate RFID and Petri Net present solutions in several different areas such as quality management of a process [6], logistic process modelling [7], healthcare [8], control design of flexible cells of manufacturing system [9], monitoring and control of assembly and disassembly systems [10], material management among others [5]. These applications have a low-level connection between Petri Net and RFID, and they focused on the creation of the Petri Net marking generation based on the reading of the tags.

The PNRD [11] provides a formal data structure to tagged objects, which defines and introduces the process activity into a RFID disperse database. In this way, the tag stores its own Petri net (incident matrix and object actual state), and readers have the control vector associated with the reading activity and another conditioning sentence allowing the automatic object Petri net next state calculation, as well as updating its own state vector after the calculation. Since the tag refers to a single object, the PNRD must be a safe Petri net, and the calculation of the next state must be a unitary vector. Any result other than a unit vector identifies an inconsistency in the process of tag, and it is viewed as an exception. A software called DEMIS (Distributed Environment Manufacturing Information System) performs the next PNRD state calculation. The RETIM (Real Time Item Monitoring) software graphically displays the tag corresponding Petri net model and actual state in real time.

Another RFID feature is the object localization through an Indoor Positioning System (IPS), which determines the position of an object in an indoor environment. Recent IPS techniques based on RFID generally uses the Received Signal Strength (RSS) information to estimate the location of a tagged object [12]. IPSs can be used for different applications that can range from detection and tracking of items, production assistance, and process monitoring [13]. With the development of automation and control, different industries rely more on IPSs for their operations such as robotic guidance, industrial robots, robot cooperation, and smart factories [14].

The PNRD approach integrated with IPS increase the value of RFID technology and it provides an example, in which the RFID implementation can be seen as a positioning sensor (IPS), and as an automatic data and process capture tool. In this context the RFID technology cannot only reach intelligent product requirement, as well as, it can be useful as IPS tool inside the smart factory approach, too. This work proposes to integrate PNRD and IPS so that the data and process is stored in the tag data memory, as well as it is possible to obtain the tag position by the IPS. A case study of a mobile robot with a passive tag is presented, in which the mobile robot changes its movement direction depending on the vehicle's distance from the reader antenna. The PNRD stores the Petri net incidence matrix as well as the robot actual state. This state changes according to the calculated distance. The vehicle moves away from the reader antenna whenever the distance is less than 35 cm and it approaches the reader antenna whenever the distance is greater than 70 cm.

This article presents Petri Net and RFID review in section 2. Section 3 shows PNRD and IPS integration purpose. Section 4 describes the mobile robot imple-

mentation. Conclusions are presented in section 5, followed by acknowledgments and references.

2 Petri Net, RFID and IPS

2.1 Petri Net and RFID Integration

On one hand, PNs provide the formal foundation formal modeling concurrency and synchronization [15]. PNs have been successfully used to model, control, and analysis discrete event dynamic systems that are characterized by concurrency or parallelism, asynchrony, deadlocks, conflicts and event-driven processes [2]. On the other hand, RFID is an automatic identification and data capture (AIDC) technology with usually presented as composed by three parts, RFID tags that is connected physically to objects; RFID reader that generates an electromagnetic field to stimulates RFID tag response when it is near enough; and RFID middleware that cares about data filtering, reader management, and application connection. There are many papers integrating RFID and PN.

Chen [9] build a CPN models for different modules of FMS (Flexible Manufacturing System), to plan RFID codes rules of FMS and to develop a cell controller for RFID-based on a centralized FMS cell controller. This article provides a suggestion for mapping between color tokens of place in the CPN and the data memory of RFID tags. Petri Net model defines RFID read & write action. RFID tag data is position sensitive, and the implementation used a low-frequency reader and tag.

Sun et al. [10] proposed an assembly executive process Petri net (AEPPN) integrating Petri nets and mobile agent-based complex product assembly framework. This approach describes states of assembly as PN transitions, events in assembly executive process as PN places and mapped to RFID tags states, which are able to trigger dispatching of assembly agents and executive of assembly tasks. AEPPN is a set of places, transitions, color set, input function, output function, initial marking and time delay transition. The mapping relationship between the product set and the color set is 1:1. In each net, the amount of token with an exclusive color is 1 and only 1. RFID tag's states can be used to describe the assembly executive process state. AEPPN can acquire, delete, create or update Tag data; therefore the AEPPN is center-controlled but executed dispersedly. RFID tags can also be regarded as offline communication channels.

Lv et al. [6] developed RFID-based CPN to improve the quality of the system without sacrificing any one of the performance parameters. RFID system elements, which are connected bidirectionally with CPN, can update information promptly with real-time action. RFID tag and color token stored the information of the product in a manufacturing system, and both of them can update the status of product simultaneously. This approach combined RFID and CPN for simulation analysis. CPN simulation results can help update the RFID database, and both databases can be synchronized. This research developed the RFID-based colored Petri net to finish the accurate real-time analysis for the manufacturing

system, so as, to realize automatic abnormity handling and enhance decision making. CPN token color remains the same after transition activity if this processing developed smoothly. Otherwise, color changing indicates failure modes happening in the last transition activity. Once the color of the token changes, the reader antenna sensor receives a signal that the status of the product changed. Then the reader rewrites the stored information and sends the new data to host application. Host application feedbacks a corresponding process activity on the colored changed token, for example, the failure part needs rework by reentrant.

Zhang et al. [2] presented a real-time production performance analysis and exception state diagnosis model (PAEDM). By combining RFID, hierarchical-timed-colored Petri Nets (HTCPN) with decision tree algorithm, this paper proposes a real-time production performance analysis and exception diagnosis model. The proposed architecture relies on three modules. The first one is IoT-enabled shop-floor module that is a bridge for information communication between physical manufacturing systems and the process. The second module deals with dynamic behavior model of the manufacturing system and data capture processing. The third module corresponds to decision tree-based exception and cause diagnosis. It presented a case scenario from a collaborative company using high-frequency RFID tags and readers. There was a need for integration with CAD/CAM/CAPP systems to perform the presented case.

Guo et al. [16] proposed a timed colored Petri net simulation-based self-adaptive collaboration method for Internet of Things-enabled production-logistics systems. The method combines the schedule of token sequences in the timed colored Petri net with real-time status of key production and logistics equipment. The proposed framework is composed of three layers, namely physical layer, cyber layer and the application one, where a Timed Colored Petri Net (TCPN) model is developed to depict and control the behavior of key equipment by adjusting the schedule of token sequences. In the simulation, a personal computer, fifteen antennas, four RFID readers, and nine RFID tags were used. The RFID tags were attached to different manufacturing objects, such as machines, AGVs, and WIP. The TCPN model started running at the same time when the production and logistics were executed according to the planned time. Firstly, real-time status information of machines, AGVs, and work in process (WIP) was transmitted to the PC through RFID reader ports. Secondly, based on the collected information, the objective functions were implemented and the results were stored in Standard ML (SML) files. Thirdly, every time the cycle of the TCPN model started, the information in the SML files was updated. By loading SML files, the status of colored tokens was tuned accordingly. Comparing TCPN based self-adaptive collaboration method with an event-driven method, total waiting time reduced 28,8%, makespan decreased 16,5%, and total electricity consumption down 4%.

Jiang et al. [17] presented a Petri-Net model-driven methodology for the development, validation, and operation of a RFID-enabled decentralized FMS. A methodology to define active and passive elements was presented in order to each active resource is equipped with a reader and each passive resource

is banded with a tag. Active resources acquire the status of passive ones by analyzing the PN models; they decide the next steps by combining their own status and behavior logic.The Color Petri Net model presented two distinguished places, it means, state-place for real-time status of the equipment, and port-place for an interface of workpiece and storage equipment.

It can be noticed that most of these applications have a low-level connection between Petri Net and RFID usually generated by the color token relationship with RFID tag reading, it means they focused on the creation of the Petri Net marking identification based on the reading of the tags in a centralized PN control model. In the case of Jiang et al. [17], RFID is the product database itself for operational level management; however, it is not clear how strong this connection is.

2.2 PNRD

According to [11], the PNRD - Petri Net Inside RFID Database - is a RFID data structure based on the elementary Petri Net formalism or Low-Level Petri Net (LLPN), and it can be described as a five-tuple (P, T, A, w, M_0), where P is the finite set of places, $P \neq$, T is the finite set of transitions, $T \neq$. $A \subseteq (P \times T) \bigcup (T \times P) \rightarrow \mathbb{N}$ is the set of arcs from places to transitions and from transitions to places, $w : A \rightarrow \{1\}$ is the unit weight function on the arcs, and $M_0 : P \rightarrow \{0, 1\}$ is the PN initial marking. As PNRD has a 1:1 relation with each tag, and PNRD must be a safe Petri net with only one weight function, and a unitary marking. In this approach each tag stores its own incidence matrix and state vector of a Petri net referring to the process part to which the tagged object in question participates; and each reader stores the corresponding control vector list and the triggering conditions. The PNRD operation is based on the capture of the tagId followed by the A^T or incidence matrix, and the tag state vector (M_k). The software responsible for calculating the next state finds the corresponding u_k (control vector) related to the conditioning set composed by tagId, tag state, antennaId, readerId, and other optional additional data, such as time interval, the distance among other. The calculation of the next tag state $M_k + 1$ follows (1).

$$M_{k+1} = M_k + A^T * u_k \quad k = 1 \ldots n \ . \tag{1}$$

The next tag state result must be evaluated. If the result is a unitary vector, this means that the Petri net remains elementary and safe, which is consistent with the fact that each tag has a 1: 1 relation with Petri Nets. This result is supposed in agreement with the expected process flow, allowing the record of the M_{k+1} in the tag memory as new tag state. Otherwise, the Petri net is no longer safe, indicating an abnormality in the expected follow-up of the process, which can generate a real-time warning signal. It is able to monitor the process of each tag individually. Even flexible processes can be stored, giving to the tagged object the ability to follow different paths as long as properly planned and modeled previously. One of the possible problems during the execution is the appearance

of conflicts. Conflicts occur when the same antenna/ reader is associated with more than one transition relative to the same tagId, tag state, and additional data. A decision algorithm can be applied to choose what transition should be triggered in order to solve the conflict and more details was presented in [11]. Hence, PNRD is based on a previously modeled system, and it is able to check whether the desirable model is followed or not.

It is possible to point out that in the PNRD approach there is a strong connection between RFID and Petri Net, which reduces the need for queries in external databases. In the other hand, it is evident that the PNRD approach uses an additional step of capturing data related to the incident matrix and the control vector. In this direction, the process of the tagged object must be predefined in advance. After this process modeling, an operational management system must attribute a specific PN process to each tagged object.

To explain PNRD didactically, Fig. 1 shows an aerospace product selection example. In this process, the product must be tested twice. The first selection defines whether the product can be sent to industrial Companies by $Test1Acceptance$ transition or not, it means, it must be sold as a replacement item by $Test1Rejection$ transition. The industrial supply item must be selected as aerospace supply by $Test2Rejection$ transition or military one by $Test2Acceptance$ transition. Since the PNRD is the Petri Net of the tagged object point of view and this object is one and only one, physically, it is not possible to split an object identification as an AND-split transition. In this example, there is no AND-split transition, so, the original Petri net model is identical to the PNRD model. In the PNRD model, there is a need for identifying the reader antenna associated with each transition. In this case, transitions $Test1Acceptance$ and $Test2Rejection$ are connected with Reader1 – Antennas 1/2, and other transitions have a distinguish reader antenna, for instance, $Test1Rejection$ is connected with Reader1 – Antenna 3, and $Test2Acceptance$ is connected with Reader1 – Antenna4. An initial marking is included in $AerospaceProduct$ state, as presented in Fig. 2.

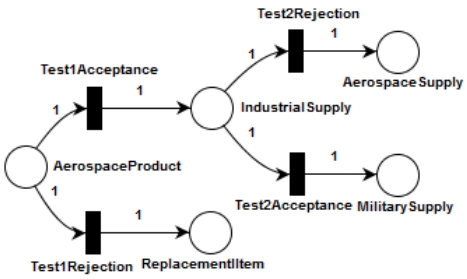


Fig. 1. The Petri Net Model of the Aerospace Product Test Process

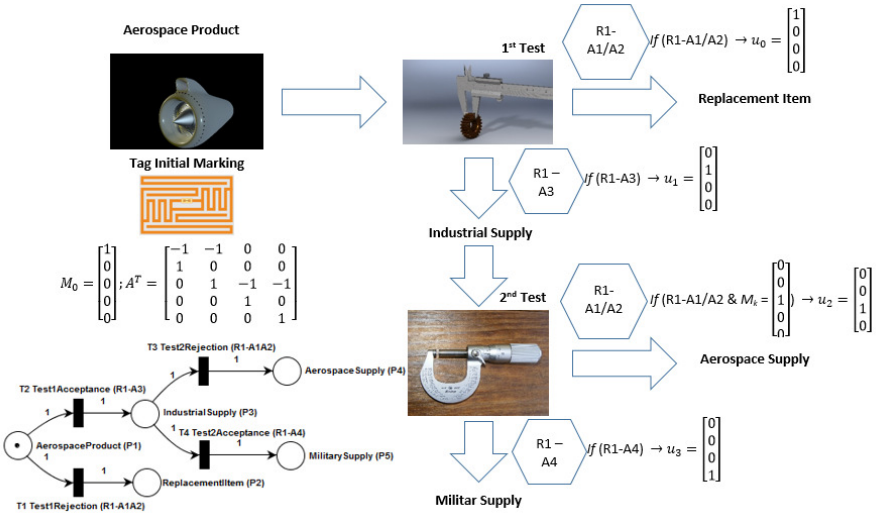


Fig. 2. The Aerospace Product Test Process Schema With PNRD Model, Initial Marking, Incident Matrix, Readers/Antennas and Control Vector Conditions

Figure 2 also presents the schema of the physical layout with one reader1 and four antennas installation. Each tagged Aerospace Product stores tagId, PNRD incident matrix, and tag state (represented by the tag initial marking M_0). Each control vector has specifics conditions as if reader1 – antennas1/2 captures a tag data, the control vector must be $[1,0,0,0]^T$ ($Test1Rejection$ activity), unless tag state is equal to $[0,0,1,0,0]^T$ where the control vector changes to $[0,0,1,0]^T$ ($Test2Rejection$ activity). This distributed data allows the automatic next state calculus during tag data capture by a specific reader. As an example, if the reader1 – antenna 1/2 captures one tag in the state M_0 ($AerospaceProductstate$), then the next state calculation result is the $ReplacementItem$ state as presented in (2).

$$M_1 = M_0 + A^T * u_1 = (0,1,0,0,0)^T \tag{2}$$

As the PNRD model must be a one-safe Petri Net, the tagged Aerospace Product cannot be in more than one state. This feature allows an automatic exception state detection. For instance, if an $AerospaceProduct$ is in the $ReplacementItem$ state and reader1 – antenna 3 is triggered, the result of the next state calculation identifies and absent of token in the $AerospaceProduct$ state, one token in the $IndustrialSupply$ state and a remaining token in the $ReplacementItem$ state (3).

$$M_2 = M_1 + A^T * u_1 = (-1,1,1,0,0)^T \tag{3}$$

DEMIS – Distributed Environment Manufacturing Information System DEMIS or Distributed Environment Manufacturing Information System is

an implementation of PNRD in software based on Java technology. The DEMIS has two modules: the PNRD core and the interfaces one. The interface module is responsible for communicating with the various devices, such as RFID readers, PLCs - Programmable Logic Controller, and the interpretation of the data sent and received. The DEMIS Core has the next state calculation algorithm or PNRD Engine; an Inference Machine with a knowledge base to solve conflicts; and configuration files (ips, port, the number of readers' antennas, control vector list and tags state pre and post conditions). Figure 3 presents, in a simplified way, the DEMIS architecture.

ReTIM – Real Time Item Monitoring Since each tag stores its own process data and process, it is possible to visualize the object operational state and process, graphically. The ReTIM software integrates the concept of process remote monitoring related with individual tagged object. After DEMIS calculates the next state, it sends a message to the ReTIM with reader/antennaId, tagId, incident matrix and tag state. Then ReTIM can graphically display the Petri Net of the object and its respective state [18]. Figure 4 shows an example of a tag data capture from DEMIS integrated with ReTIM to graphically visualize a Petri Net with four places, four transitions, and the marking in the P4 state.

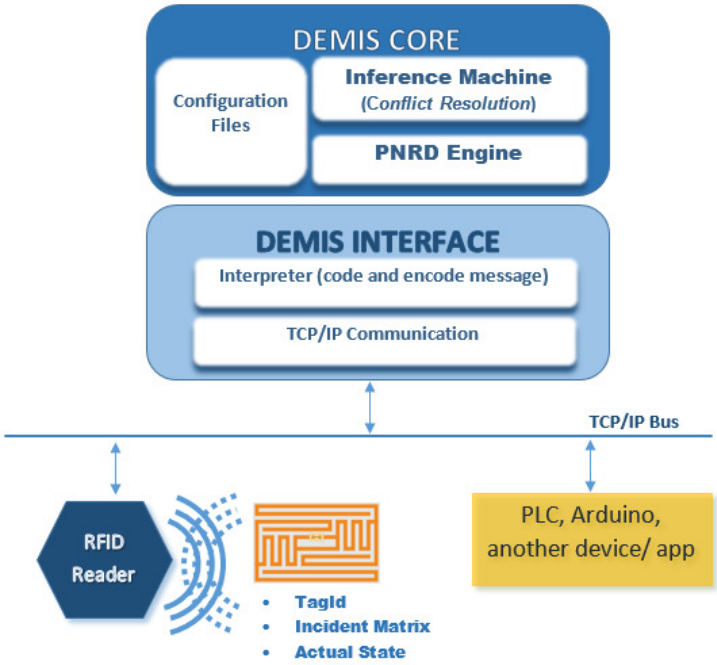


Fig. 3. DEMIS Architecture adopted from [11]

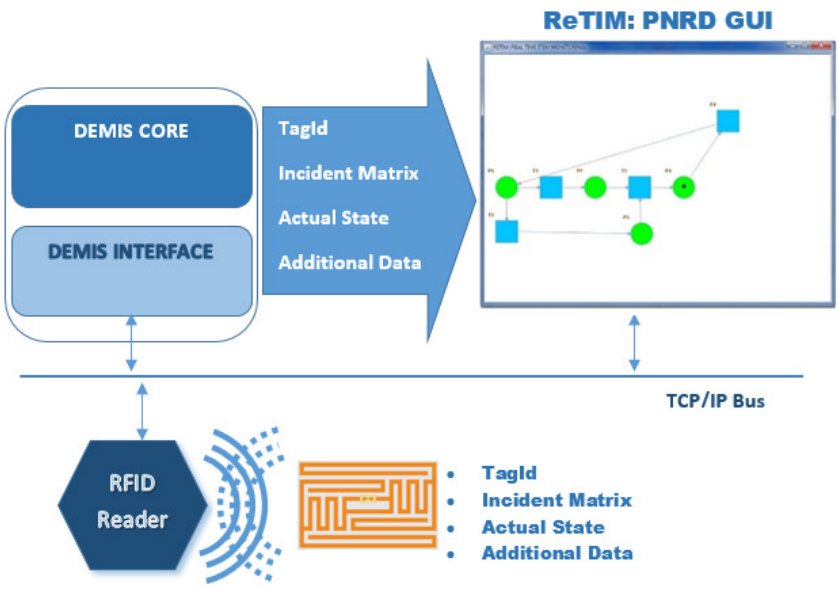


Fig. 4. Example of ReTIM interface with DEMIS

2.3 IPS (Indoor Positioning System)

There are two types of RFID indoor positioning system, i.e., reader localization, and tag localization depending on what, between reader and tag, needs to be localized. In the reader localization, the accuracy of the RFID system is highly depending on the density of tag deployment and the maximal reading ranges. In a probable localization context, a large number of RFID tags, which contain its own location information, can be deployed to cover an entire indoor environment. The disadvantage of this approach is the large number of RFID tags, which need to be applied, and prerecorded in advance with location information. Obviously, this method is more expensive and the cost increases with the increase in the number of used RFID readers [19].

Related with IPS algorithms, there are four types, it means, Time of Arrival (TOA), Time Difference of Arrival (TDOA), Angle of Arrival (AOA) and Received Signal Strength (RSS). RSS estimates the distance of an unknown node to reference node from some sets of measuring units using the attenuation of emitted signal strength. This method can only be possible with radio signals. RSS localization method could be using either a propagation model algorithm or a fingerprinting algorithm. Propagation Model Algorithm (PMA) establishes the model between RSS and the distance. Generally, the higher of the RSS values the closer from the Access Point (AP) the tagged object is. Attenuation of the received signal strength is inversely proportional to the distance from AP

in the outdoor. In contrast, it is complex in the indoor environment because of the existence of obstacles (furniture, equipment windows, doors etc.) may cause multipath propagation, such as reflection, refraction, and diffraction [14].

Indoor localization of autonomous vehicles (or mobile robots) is a challenging and lively subject because of the complexity of the indoor scenarios, the diversity of technologies involved, and the commercial and industrial interests [20] and one possible way of estimating a robot position makes use of the RFID. This subject has received a considerable attention in the last few years and in many cases, the localization system is realized by installing a reader on the robot and by providing the environment with a certain number of tags placed in known position [13, 21–23]. Applying formal methods to model robot tasks like Petri net provides a systematic approach to modeling, analysis, and design, scaling up to realistic applications, and enabling analysis of formal properties, as well as design from specifications [24].

There are also many papers about IPS based on RFID technology and the main purpose of this subsection is to present some works related to mobile robot localization.

DiGiampaolo and Martinelli [22, 23] propose a global localization system combining odometry data with RFID readings. The RFID tags are placed on the ceiling of the environment and can be detected by a mobile robot unit traveling below them. The detection of the tags is the only information used in the proposed approach (no distance or bearing to the tag is considered available), but only a small number (about one each square meter or less) of tags are used. This is possible using a suitable tag's antenna in a ultrahigh frequency band, expressly designed to obtain regular and stable RFID detection regions. A satisfactory performance is achieved, with an average position error of about 0.1 m.

Martinelli [20] proposes a global positioning system based on the received signal strength and the phase shift of UHF-RFID signals coming from a set of passive tags deployed on the ceiling of the environment together with odometry provides the position of a mobile robot. A multi-hypothesis extended and unscented Kalman filter is proposed to localize the robot and to simultaneously improve the initial estimate on the tag coordinates.

Murofushi et al. [25] and Murofushi and Tavares [26] designed a real time unidimensional indoor positioning using passive tags based on the RSS of the backscattered signal. The IPS design was based in the system calibration, and the distance estimation phase. The IPS accuracy achieved is 4.7 cm for a mobile robot moving at constant velocity.

Errington et al. [27] investigate the concept of using an array of RFID tags placed at fixed known positions to provide the initial position to the Simultaneous Localization and Mapping (SLAM) algorithm. The mobile vehicle has a RFID tag reader coupled to it and the antenna is used to detect the tags. The application of interest here involves determining the initial position of a stationary vehicle in an underground mine using an array of RFID tags placed at known positions to provide the initial position of the vehicle. The results suggest that

RFID-based positioning, using the Least Square approach, has the potential to provide relatively accurate and low-cost initial position estimation.

3 PNRD and IPS Integration Proposal

The proposal of this article relies on increasing RFID and Petri Net operational potential application based on IoT approach, it means, to use RFID system as a process aware based on PNRD approach integrated with an IPS.

In this sense, RFID IPS sensor must become a pre or post condition enabling or inhibiting one or more PNRD transitions. This arrangement changes PNRD from ordinary to a high-level PN. It is necessary to complement the PNRD formalism with the pseudo-box concept, which denotes an observable condition that is not controlled by the modeled PN; and disabling arc [28].

As presented in [28], pseudo-box is a hierarchical resource embedded in Petri Net, that is, elements that only propagate information and preserve the marking in its original place. It could be an enabling gate, that is, one that sends information if is marked, or an inhibitor gate, if propagates information when is not marked. Thus, these gates must always have an original place in Petri Net graph, a special place called pseudo-box.

Pseudo-boxes denotes an observable condition that is not controlled by the modeled system. During the course of the modeling, pseudo-boxes could also stand for control information external to the hierarchical components and could be collapsed when components are put together. Thus, pseudo-boxes must be considered in the structure of the net but should not affect its properties or the rank of the incidence matrix.

This high-level Petri Net is a five-tuple (L, T, A, w, M_0), where L is the finite set of places and pseudo-box, $L = B \bigcup P$, $L \neq$, T is the finite set of transitions; $T \neq$, $A \subseteq (L \times T) \bigcup (L \times P) \rightarrow \mathbb{N}$ is the set of arcs from places or pseudo-box to transitions and from transitions to places or pseudo-box; $w : A \rightarrow \{1\}$ is the unit weight function on the arcs; and $M_0 : P \rightarrow \{0, 1\}$ is the PN initial marking.

Hence, PNRD extended to distance is the original PNRD with pseudo-box and disabling arc. This new information is calculated and storage at the reader and Fig. 5 shows its correspondent sequence diagram. If the precondition response identifies a required distance range, this generates a new operation in order to determine tag distance and check if it is inside transition disabling the rule. If so, next state calculation is realized. In this case, the internal application runs PNRD and IPS algorithm.

Next section presents the case study of a mobile robot position control using PNRD extended to distance approach.

4 Case Study: Implementation of PNRD and IPS in Mobile Robot Position Control

The case study presented in this work is about a mobile robot controlled by the PNRD. The vehicle moves forward or backward from 35 to 70 cm in an oscillating

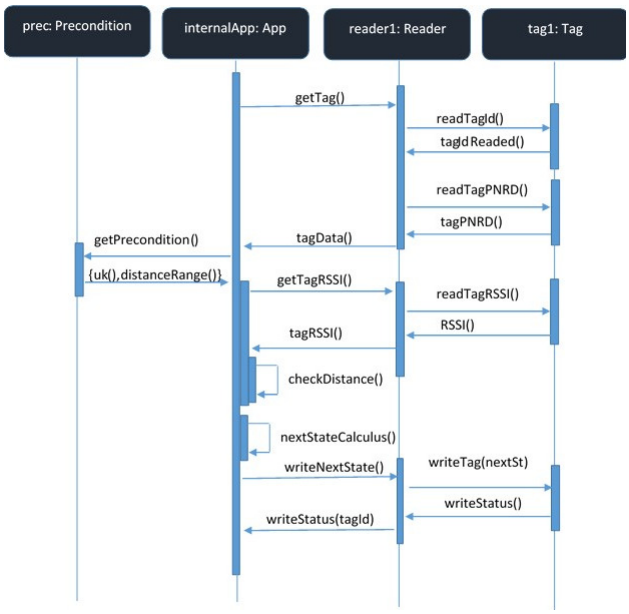


Fig. 5. Sequence diagram of proposed PNRD extended to distance

cycle. Figure 6 shows a scheme of the mobile robot position control. The mobile robot uses two stepper motors, it has a short dipole tag attached, and an Arduino Uno R3 controls it. The reader is a reader M6e micro with a monostatic antenna. DEMIS and IPS were implemented in java programming language in an Intel core I5 750, 2.66GHz, 8 GB RAM DDR3 in Windows 10 Pro 64 bit platform. The algorithm takes about 200ms for each position estimation (data processing operation) and the vehicle was programmed to move at a constant velocity of 100 mm/s. Therefore, the mobile robot positioning error is lower than the IPS accuracy of about 57 mm. Therefore, the identification of the position of the robot by the IPS can be identified as in real time.

4.1 Mobile robot position control Petri net model

Figure 7 presents mobile robot Petri net model with places (white circle) and pseudo-box (gray circles), transitions, arcs and disabling arcs. There are four places, $InitialMarking$ ($P1$), $MobileRobotStandBy$ ($P2$), $ForwardMovement$ ($P3$) and $BackwardMovement$ ($P4$) and five pseudo-box $DistanceMeasurement$ ($Ps1$), $FwdMovMessage$ ($Ps2$), $BckMovMessage$ ($Ps3$), $StopFwdMovMessage$ ($Ps4$) and $StopBckMovMessage$ ($Ps5$). Places define mobile robot dynamic behavior and pseudo-box deals with external sensing ($Ps1$) and command messages ($Ps2$ to $Ps5$). The estimated distance "d" is calculated, and, depending on "d" value, a transition may be triggered. For instance, when the vehicle is in the $P2$ state, $Ps1$ receives "d" from IPS; and, if "d" is less or equal to 50cm, $T2$ triggers

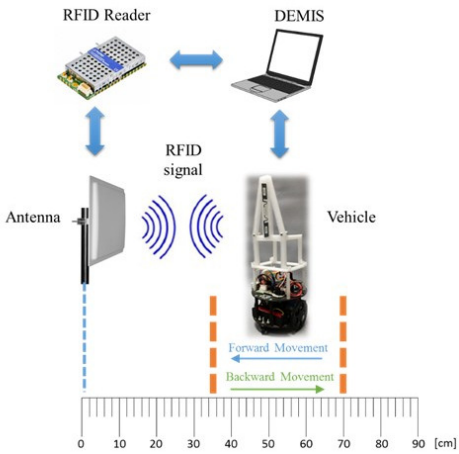


Fig. 6. Scheme of the mobile robot position control

changing the vehicle state from $P2$ to $P3$ and $Ps2$ sends a message related to the triggered transition. Each place, pseudo-box, and transition have a corresponding with RFID system. Table 1 describes places and pseudo-box, and Table 2, transition. In the mobile robot example, pseudo-boxes are applied as the robot distance control as IPS interface. It can be noticed that the configuration file stores distance setup to be reached by the robot and represents robot control pre-conditions to change its own direction.

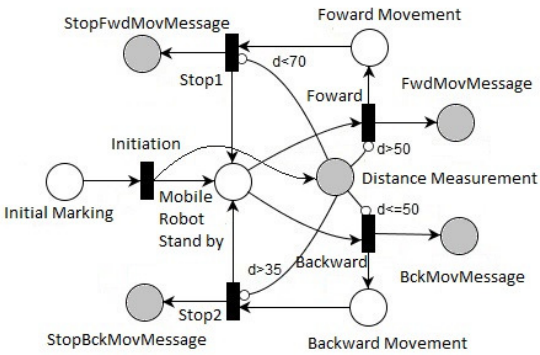


Fig. 7. Mobile robot Petri net model

Table 1. Description of places and pseudo-box

Meaning	Corresponding RFID System
Initial marking	Tag(incid. matrix and state vector)
Mobile robot stand by	Tag (incid. matrix and state vector)
Forward movement	Tag (incid. matrix and state vector)
Backward movement	Tag (incid. matrix and state vector)
Distance measurement	Reader (DEMIS Position algorithm)
Forward movement message	Reader (DEMIS Core Conf. File)
Backward movement message	Reader (DEMIS Core Conf. File)
Stop forward movement message	Reader (DEMIS Core Conf. File)
Stop backward movement message	Reader (DEMIS Core Conf. File)

Table 2. Description of transitions

#	Meaning	Corresponding RFID System
T1	Position control initiation	Reader (DEMIS Core Conf. File)
T2	Distance less or equal than 50	Reader (DEMIS Core Conf. File)
T3	Distance more than 50	Reader (DEMIS Core Conf. File)
T4	Distance more or equal 70	Reader (DEMIS Core Conf. File)
T5	Distance less or equal 35	Reader (DEMIS Core Conf. File)

4.2 IPS deployment

The IPS must be calibrated first so that an expression of the distance in function of RSS is estimated. Then the estimated distance can be calculated. The calibration is made by collecting 500 samples of RSS values every 3 cm in the range from 30 to 90 cm, measured from the antenna. Since the antenna varies the frequency of the emitted signal, in a range of 50 distinct frequencies between 902 and 928 MHz, it is also important to associate the signal frequency with the respective RSS value and distance.

After data collection, a second-degree calibration curve is fitted for each individual frequency, associating a RSS value with a distance in centimeters. Then, utilizing those curves, a mean distance is computed for each position (and the confidence interval for the each location was estimated for a significance of 5%). Equation (4) shows the calibration expression for the distance in function of the RSS, where c_0, c_1, and c_2 are constantly obtained from calibration process, and they depend on signal frequency.

$$Distance = \frac{-c_1 - \sqrt{{c_1}^2 - 4 * c_2 * (c_o - rss)}}{2 * c_2} \tag{4}$$

Figure 8 shows a graphic of the experimental data and adjusted curve. The real distance curve is, evidently, non-linear. This is due to electromagnetic waves

present in the environment which may cause an interference in the RSS values, such as, reflection; another reason is the low resolution of the RSS reader (1 dBm) [27].

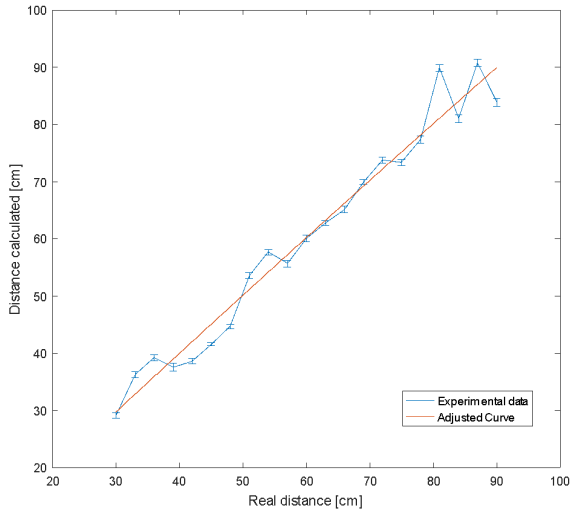


Fig. 8. Experimental data and adjusted curve graph

4.3 PNRD Extended to Distance Implementation

This case study requires a specific configuration file, which identifies state transition depending on mobile robot state and antenna distance. Figure 9 presents configuration file code, and, it can be notice the transition 1 has no distance preconditioning, only state 1, the initial marking. As the mobile robot is connected physically by USB port, Java communication is "serial" type. The transition 2 has a distance pre-condition, it means, this transition fires only if the distance is more than 50. Transitions 3 to 5 are similar with a different distance requirement. As the distance range is unique to each transition, there's no need of state identification for transition 2 to 5 in order to avoid conflict. Each "distance" label requires the petri Net pseudobox Ps1 external sensing. The "outputType" label starts the petri Net pseudobox Ps2 to Ps5 external communication.

The Fig. 10 shows DEMIS process log example when mobile vehicle is in state P2 with distance of 44 cm. As this distance is less than 50 cm, the transition T2 fires, it sends the message "2" to the mobile robot, changing mobile robot state to P3.

Figure 11 (a) and (b) presents RETIM graphical example of this state changing. Depending on the mobile robot state, it changes its own direction (move for-

```
[{"state": 1,"transition": 1,"outputType": "serial",},
{{"distanceMin": 50,"transition": 2,"outputType":
"serial",},
{"distanceMax": 50,"transition": 3,"outputType":
"serial",},
{"distanceMax": 35,"transition": 4,"outputType":
"serial",},
{"distanceMin": 70,"transition": 5,"outputType":
"serial",}]
```

Fig. 9. Mobile robot position control configuration file

ward or backward) according to the flowchart presented in Fig. 12. For instance, if the transition T2 triggers, the mobile robot moves forward. This logic control is implemented in the mobile robot Arduino. Figure 13 shows an implementation site photo.

5 Conclusion

Increase Petri net and shop floor integration, higher the opportunity to develop a complete methodology of discrete event system design, model checking, and deployment.

IoT changes to control system paradigm from centralized to distribute one. This new paradigm requires new implementation approaches in order to deal with micro processed operational level. In this direction, RFID is a cornerstone technology of IoT [2]. In a distributed environment, there is a need for a distributed modeling technique. Petri nets are commonly viewed as modeling and controlling tool; but, in control field, Petri net is usually applied as system dynamic model. PNRD is a method that fits distributed modeling technique requirement, splitting Petri net structure and storing it in RFID components readers and tags. Several examples show RFID and Petri net integration, but most of them rely on a centralized model in the control level, far away from the sensor itself [2, 6, 9, 10, 16]. Jiang et al [17] stores Petri net model in the sensor level of RFID tag, although this model is stored completely, it remains centralized.

The RFID technology is more than a distributed database, and this paper presents an IPS based on RFID RSS signal. Most of IPS research fixes signal frequency [20, 22, 23, 27]. In regular RFID application, readers have a frequency range to generate more than one communication channel.

This article presents an extended PNRD integrated with an IPS application in mobile robot positioning control. In this context, there is a need to deal with external communication between the reader and the mobile vehicle; transition triggering preconditions; and a frequency range of RFID signal. To reach these requirements, the IPS deals with the whole reader frequency range; and the PNRD approach requires a high-level Petri net structure with pseudo-box and disabling arcs. To implement this model in RFID system, pseudo-box and disabling arcs are stored in the reader configuration file; and the IPS algorithm is

```
Listening Port: 7070
----------------------Reading
tagId-----------------------------------------Reading
tagData--------------------Processing tag data:
Number of States: 4
Number of Transition: 5
Incident Matrix:
-1  0  0  0  0
 1 -1 -1  1  1
 0  1  0 -1  0
 0  0  1  0 -1
Actual State: P2
-------------------Configuration File -------------------
Reading configuration file pre-condition…
-------------------Getting Preconditions----------------
Distance requesting:
RSSI: -36
Calculated distance: 44.24500617599104 [cm]
----------------Control Vector definition----------
Transition T2 pre-condition satisfied!
--------------------Next state calculation--------------
Transition T2 fired!
Sending message: 2
New state: P3
Writing new state in tag…
Writing status: Ok
```

Fig. 10. DEMIS process logging next state calculation example from state P2 to state P3

embedded in PNRD engine. An example is presented where the PNRD is the control logic algorithm, and the integrated IPS is the positioning measurement of the vehicle at the operational level. This paper demonstrates the feasibility of integration between Petri nets and RFID technology through PNRD and IPS. This approach increases RFID value generation, creating opportunities for new improvement in several areas.

The PNRD approach must be extended to time with time Petri nets [28], continuous process with continuous Petri nets [29], process mining [30], color Petri nets [15], and the Petri net next state writing process may have technical issues, which means that an internal communication procedure must deal with cases of recording problems. RFID hardware must be improved to find a more reliable and accurate positioning results, a new position algorithm using RSS and phase signal, treating the full range of radio frequency, is demanded.

6 Acknowledgement

CAPES, CNPQ, FAPEMIG and UFU supported this research.

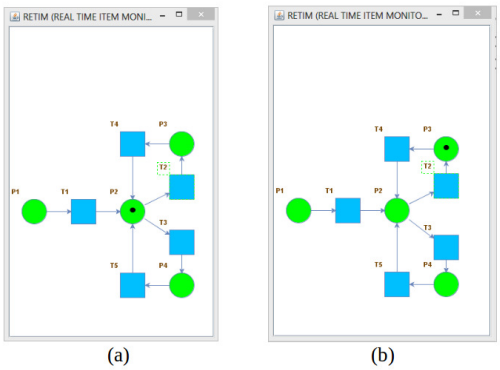


Fig. 11. RETIM graphical example mobile robot state changing from P2 (a) to P3 (b)

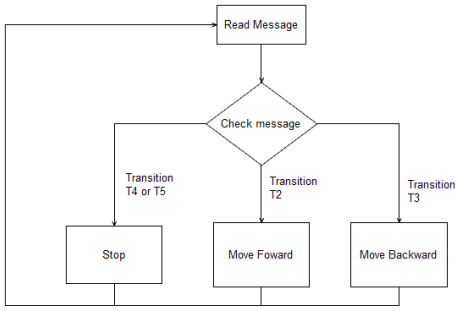


Fig. 12. Mobile robot Arduino code flowchart

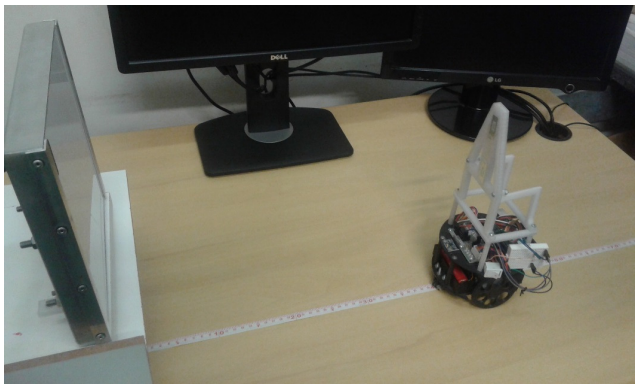


Fig. 13. Implemented mobile robot in front of the reader antenna

References

1. GTAI – German and Trade Investment. (2016, Nov. 6): Industrie 4.0 – Smart Manufacturing for the Future, Online. Available:https://www.gtai.de/GTAI/Content/EN/Invest/_SharedDocs/Downloads/GTAI/Brochures/Industries/industrie4.0-smart-manufacturing-for-the-future-enpdf?v=8
2. Zhang, Y., Wang, W., Wu, N.: IoT-Enabled Real-Time Production Performance Analysys in Exception Diagnosis Model. IEEE Transaction on Automation Science and Engineering, Vol 13, no 3, pp. 1318-1332, July 2016.
3. Borgia, E.: The Internet of Things vision: Key features, applications and open issues. Computer Communication, 54, pp. 1-31, 2014. Available at http://www.sciencedirect.com/science/article/pii/S0140366414003168.
4. McFarlane, D., Sarma,, S., Chirn, J. L., Wong, C. Y., Ashton, K.: The intelligent product in manufacturing control. In Proc. 15th IFAC World Congress, 2002.
5. Ngai, E.W.T., Moon, K.K.L., Riggins, F.J., Yi, C.Y.: RFID research: An academic literature review (1995-2005) and future research directions. International Journal Production Economics, 112 (2008), pp. 510-520.
6. Yaquiong Lv, Lee, C.K.M., Chan, H.K., Ip: RFID-based colored Petri net applied for quality monitoring in a manufacturing system. Int J Adv Manuf Technol 60, pp 225–236, 2012.
7. Oberwies, A., Zhang, H.: Modelling and facilitating RFID-based collaborative logistics processes. International Journal of Organizational Design and Engineering, Vol. 2, No. 1, 2012.
8. Fernandez-Llatas, C., Lizondo, A., Monton, E., Benidi, J.M., Traver, V.: Process-Mining Methodology forHealth Process TrackingUsingReal-Time IndoorLocation Systems. Sensors, 15, pp. 29821-29840, 2015.
9. Chen, K.: Cell controller design for RFID based flexible manufacturing systems. International Journal of Computer Integrated Manufacturing, Vol. 25, No. 1, January 2012, pp. 35-50, Taylor & Francis.
10. Sun, H., Chang, Z., Mo, R.: Monitoring and controlling the complex product assembly executive process via mobile agent and RFID tags. Assembly Automation, Vol 29, Issue 3, pp. 263-271, Emerald Insight, 2009.
11. Tavares, J.J.P.Z.S., Saraiva, T.A., Elementary Petri Nets inside RFID Database (PNRD). Journal International Journal of Production Research. Vol. 48, 2010 - Issue 9: RFID Technology and Applications in Production and Supply Chain Management.
12. Zhou, J., Zhang, H. Mo, L.: Two-dimension localization of passive RFID tags using AOA estimation. In Proc. IEEE Instrumentation and Measurement Technology Conf. (I2MTC), pp. 1-5, 2011.
13. Zhou, J., Shi, J., "RFID localization algorithms and applications, a review". Journal of Intelligent Manufacturing, pages 1–13, 2008.
14. Al-Ammar, M. A., Alhadhrami, S., Al-Salman, A., Alarifi, A., Al-Khalifa, H. S., Alnafessah, A., Alsaleh, M.: Comparative survey of indoor positioning technologies, techniques, and algorithms. In Proc. International Conference on Cyberworlds (CW), pp. 245-252, 2014.
15. Jensen, K., Kristensen, L.M.: "Colored Petri Nets: A graphical language for formal modeling and validation of concurrent systems". Communication of the ACM, Vol. 58, no. 6, June 2015.

16. 16. Guo, Z., Zhang, Y., Zhao, X., Song, Z. "A Timed Colored Petri Net Simulation-Based Self-Adaptive Collaboration Method for Production-Logistics Systems". Applied Sciences, 2017, 7, 235. Doi:10.3390/app7030235.
17. Jiang, Z., E, M., Liu, Y., Liu, J., Li, Y.: "Study of manufacturing resource perception and process control of a radio-frequency-identification-enabled decentralized flexible manufacturing system". Advances in Mechanical Engineering, 2017, Vol. 9(I) pp. 1-12.
18. Molina, F. S. ; Marco, W. P. ; Sousa, A. R. ; Tavares, J. J. Z. S. . "Algoritmo para visualização de processos de manufatura baseado na PNRD". In: Congresso Brasileiro de Automática, 2012, Campina Grande. XIX CBA - Congresso Brasileiro de Automática, 2012. Available at www.eletrica.ufpr.br/anais/cba/2012/Artigos/99831.pdf. Accessed in April 2, 2017.
19. Mainetti, L., Patrono, L., Sergi, I.: A survey on indoor positioning systems. In Proc. 22nd International Conference of Software, Telecommunications and Computer Networks (SoftCOM), pp. 111-120, 2014.
20. Martinelli, F.: A Robot Localization System Combining RSSI and Phase Shift in UHF-RFID Signals. 2015.
21. Shen, J., Wang, A., Wang, C., Ren, Y., Sun, X.: A RFID Based Localization Algorithm for Wireless Sensor Networks. Cloud Computing and Security. Vol. 10040 of the series LNCS pp 275-285, 2016.
22. DiGiampaolo, E., Martinelli, F.: "A Passive UHF-RFID System for the Localization of an Indoor Autonomous Vehicle" Ieee Transactions On Industrial Electronics, [s.l.], v. 59, n. 10, p.3961-3970, oct. 2012. Institute of Electrical & Electronics Engineers (IEEE).
23. DiGiampaolo, E., Martinelli, F.: "Mobile Robot Localization Using the Phase of Passive UHF RFID Signals" IEEE Transactions On Industrial Electronics, [s.l.], v. 61, n. 1, p.365-376, Jan. 2014. Institute of Electrical & Electronics Engineers (IEEE).
24. Costelha, H., Lima, P.: Robot task plan representation by Petri nets: modeling, identification, analysis and execution. Autonomous Robots, Springer, Volume 33, Issue 4, pp 337-360, 2012 Springer Nature.
25. Murofushi, R. H., Gonçalves, R. F., Sousa, A. R., Tavares, J. J. P. Z. S., "Indoor Positioning System Based on the RSSI Using Passive Tags." XIII Latin American Robotics Symposium and IV Brazilian Robotics Symposium (lars/sbr), [s.l.], p.323-327, out. 2016. Institute of Electrical and Electronics Engineers (IEEE).
26. Murofushi, R. H., Tavares, J. J. P. Z. S., "Towards Fourth Industrial Revolution Impact: Smart Product Based on RFID Technology", to be published in April 2017. IEEE Instrumentation and Measurement Magazine.
27. A. Errignton, F. C. Angus, B. L. F. Daku, A. F. Prugger, "Initial Position Estimation Using RFID Tags: A Least-Squares Approach". Ieee Trans. Instrum. Meas., [s.l.], v. 59, n. 11, p.2863-2869, Nov. 2010. Institute of Electrical & Electronics Engineers (IEEE).
28. Silva J.R., del Foyo P.M.G. "Timed Petri Nets". Pawel Pawlewski (Ed.), Petri Nets: Manufacturing and Computer Science, InTech (2012), pp. 359-378.
29. Julvez, J, Vazquez CR, Mahulea C, Silva M. "Continuous Petri Nets: Controllability and Control" in Control of Discrete-Event Systems London U.K.: Springer vol. 433 pp. 407-428 2013. DOI: 10.1007/978-1-4471-4276-8.
30. Van der Aalst, W. "Process mining" Communication of ACM, Vol. 55, n. 8, August 2012, pp. 76, 83. DOI: 10.1145/2240236.

Application of Model-based Testing on a Quorum-based Distributed Storage

Rui Wang[1], Lars Michael Kristensen[1],
Hein Meling[2], Volker Stolz[1]

[1] Department of Computing, Mathematics, and Physics
Western Norway University of Applied Sciences
Email: {rwa@hvl.no,lmkr@hvl.no,vsto@hvl.no}
[2] Department of Electrical Engineering and Computer Science
University of Stavanger, Email: {hein.meling@uis.no}

Abstract. Data replication is a central mechanism for the engineering of fault-tolerant distributed systems, and is used in the realization of most cloud computing services. This paper explores the use of Coloured Petri Nets (CPNs) for model-based testing of quorum-based distributed systems. We have used model-based testing to validate a distributed storage implemented using the Go language and the Gorums framework. We show how a CPN model of a single-writer, multi-reader distributed storage system can be used to obtain both unit test cases for the quorum logic functions, and system level test cases consisting of read and write quorum calls to the storage. The CPN model is also used to obtain the test oracles against which the result of running a test case can be compared. Our experimental results show that we obtain 100 % code coverage for the quorum functions, 84 % coverage on the quorum calls, and 40 % coverage on the Gorums framework.

1 Introduction

Distributed systems serve millions of users in many important applications and domains. However, such complex systems are known to be difficult to implement correctly because they must cope with concurrency, failures, and a lot of other challenges [9]. Thus, when designing and implementing distributed systems, it is important to ensure correctness and fault-tolerance. Distributed systems can rely on a quorum system to achieve fault-tolerance, yet it remains challenging to implement fault-tolerance correctly. Therefore, the use of testing techniques can help to detect bugs and to improve the correctness of such systems.

One promising testing approach is *model-based testing* (MBT) [18]. MBT is a paradigm based on the idea of using models of a system under test (SUT) and its environment to generate test cases for the system. The goal of MBT is validation and error-detection aimed at finding observable differences between the behavior of the implementation and the intended behavior of the SUT. A test case consists of test input and expected output and can be executed on the SUT. Typically, MBT involves: (a) build models of the SUT from informal

requirements; (b) define test selection criteria for guiding the generation of test cases and the corresponding test oracle representing the ground-truth; (c) generate and run test cases; (d) compare the output from test case execution with the expected result from the test oracle. The component that performs (c) and (d) is known as a *test adaptor* and it uses a *test oracle* to determine whether a test has passed or failed.

The contribution of this paper is to investigate the use of Coloured Petri Nets (CPNs) [8] for model-based testing applied to quorum-based distributed systems [20]. Quorum systems are fundamental to building fault-tolerant distributed systems, and recently the Gorums framework [13] has been developed to ease the implementation of quorum-based distributed systems. The Gorums framework constitutes a distributed middleware that hides the complexity in implementing the communication, synchronization, message processing, and error handling between the protocol entities. Our long-term research goal is to validate the Go implementation of the Gorums framework using MBT. As a first step towards this goal, we consider a Gorums-based implementation of a simple single-writer, multi-reader distributed storage.

The storage system is implemented with a read and a write *quorum call*, which clients can use to access the distributed storage. The distributed storage may return multiple replies to a quorum call. To simplify client access to the storage, Gorums uses a user-defined *quorum function* to coalesce the different replies into a single reply that can then be returned to the client. For this particular storage system, we use a majority quorum. By developing a CPN model of such a distributed storage, we are able to generate test cases consisting of read and write quorum calls that test the Gorums framework implementation.

CPNs have been widely used for modeling and verifying models of distributed systems spanning domains such as workflow systems, communication protocols, and distributed algorithms [11]. Recently, work has also been done on automated code generation allowing an implementation of the modeled systems to be obtained [10]. Comprehensive testing of an implementation is, however, an equally important task in the engineering of distributed systems, independently of how the implementation has been obtained. This also applies in the case of automated code generation, as it is seldom the case that the correctness of the model-to-text transformations and their implementation can be formally proved.

The rest of this paper is organized as follows. §2 introduces quorum-based system and the Gorums framework, and §3 describes the Gorums-based distributed storage which constitutes our system under test. §4 presents the constructed CPN model for test case generation, and §5 shows how state-spaces and simulation can be used to obtain test cases and test oracles. In §6 we present the Go implementation of our test adapter and how it is connected to the Gorums implementation of the distributed storage in order to execute the test cases. We also report on the experimental results. §7 presents related work, and in §8 we sum up conclusions and presented directions for future work. The reader is assumed to be familiar with the basic concepts of high-level Petri Nets.

2 Quorum-based Distributed Systems and Gorums

In this section we describe Gorums [13], a framework for implementing quorum-based distributed systems. We have used Gorums to implement the distributed storage system that enables us to test the implementation of the framework.

Gorums is a library whose goal is to alleviate the development effort for building advanced distributed algorithms, such as Paxos [12] and distributed storage [1]. These algorithms are commonly used to implement replicated services, and they rely on a quorum system [20] to achieve fault tolerance. That is, to access the replicated state, a process only needs to contact a quorum, e.g. a majority of the processes. In this way, a system can provide service despite the failure of individual processes. However, communicating with and handling replies from sets of processes often complicate the protocol implementations.

To reduce this complexity, Gorums provides two core abstractions: (a) a flexible and simple quorum call abstraction, which is used to communicate with a set of processes and to collect their responses, and (b) a quorum function abstraction which is used to process responses. These abstractions can help to simplify the main control flow of protocol implementations.

Fig. 1 illustrates the interplay between the main abstractions provided by Gorums. Gorums consists of a runtime library and code generator that extends the gRPC [5] remote procedure call library from Google. Specifically, Gorums allows clients to invoke a quorum call, i.e. a set of RPCs, on a group of servers, and to collect their replies. The replies are processed by a quorum function to determine if a quorum has been received. Note that the quorum function is invoked every time a new reply is received at the client, to evaluate whether or not the received set of replies constitutes a quorum.

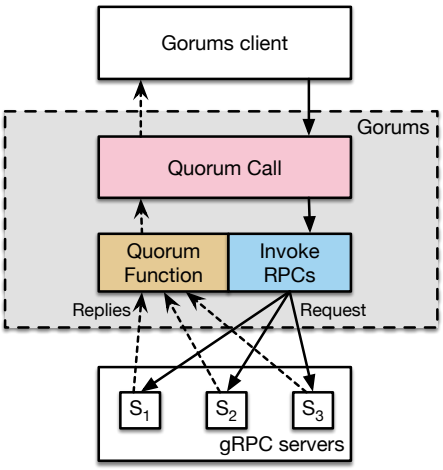


Fig. 1. Overview of Gorums abstractions.

With Gorums, developers can specify several RPC service methods using `protobuf` [6], and from this specification, Gorums' code generator will produce code to facilitate quorum calls and collection of replies. However, each RPC/quorum call method must provide a user-defined quorum function that Gorums will invoke to determine if a quorum has been received for that specific quorum call. In addition, the quorum function will also provide a single reply value, based on a coalescing of the received reply values from the different server replicas. This coalesced reply value is then returned to the client as the result of its quorum call. That is, the invoking client does not see the individual replies.

Our goal in this paper is to provide a framework for generating test cases to verify the correctness of the Gorums implementation itself in addition to different quorum function and quorum call implementations for specific use of the framework. The quorum functions for a specific protocol implementation must follow a well-defined interface generated by Gorums. These only require a set of reply values as input and a return of a single reply value together with a boolean quorum decision. Hence, quorum functions can easily be tested using unit tests. However, some quorum functions involve complex logic, and their input and output domains may be large, and so generating test cases from a model, provide significant benefit to verify correctness. A quorum call is implemented by a set of RPCs, invoked at different servers, and so must consider different interleavings due to invocations by different clients. Hence, using model-based testing we can produce sequences of interleavings aimed at finding bugs in the server-side implementations of the RPC methods and also in the Gorums runtime system.

3 System Under Test: Gorums and Distributed Storage

We have implemented a distributed storage system, with a single writer and multiple readers. The storage system is replicated for fault-tolerance, and is implemented using Gorums. To test this storage implementation, we have designed a corresponding CPN model that we use to generate test cases (see §4). In this section, we describe the different components of the distributed storage.

As with any RPC library, Gorums also requires that the server implements the methods specified in the service interface. For our distributed storage, we have implemented two server-side methods: Read() and Write(). These can be invoked as quorum calls from storage clients, to read/write the state of the storage. In our current implementation, we allow only a single writer client, but any number of clients can read the state of the storage. A client reading from the storage may observe different replies returned by the different server replicas, since the read may be interleaved with one or more writes generated by the writer client.

To allow a reader to pick the correct reply value to return from a quorum call, each server also maintains a timestamp that is incremented for each new Write(). That is, the reader will always return the value associated with the reply with the highest timestamp. Thus, to implement the reader client using Gorums, we can simply implement a user-defined ReadQF quorum function for

the Read() quorum call as shown in Algorithm 1. As this code illustrates, a set of replies from the different servers are coalesced into a single reply, the one with the highest timestamp, that can then be returned from the quorum call.

The user-defined quorum functions are implemented as methods on an object of type QuorumSpec, named *qs* in Algorithm 1. This object holds information about the quorum size, such as ReadQSize, and other parameters used by the quorum functions. This *qs* object must satisfy an interface generated by Gorums' code generator. In Algorithm 1, ReadQSize is used to determine if enough replies have been received to search for the reply with the highest timestamp.

Algorithm 1 Read quorum function

```
1: func (qs QuorumSpec) ReadQF(replies []ReadReply)
2:     if len(replies) < qs.ReadQSize then                    ▷ read quorum size
3:         return nil, false                                  ▷ no quorum yet, await more replies
4:     highest := ⊥                                           ▷ reply with highest timestamp seen
5:     for r := range replies do
6:         if r.Timestamp ≥ highest.Timestamp then
7:             highest := r
8:     return highest, true                                   ▷ found quorum
```

4 CPN Testing Model for the Distributed Storage

In this section, we describe the CPN model of our test framework for the distributed storage presented in §3. We model the entire system, parametrized by a number of clients and servers. Some key features of the model are the use of colored tokens for distinguishing multiple incoming and outgoing messages, and the quorum specification based on the numbers of replies received so far.

Fig. 2 shows the top-most module of the CPN model developed in order to generate test cases for the distributed storage implementation. The substitution transition Clients represents the clients (users) of the distributed storage system while Servers represent the servers. The places ClientToServer and ServerToClient are used for modeling the exchange of messages between the clients and the servers. The CPN model has been constructed in a folded manner so that the number of servers is a parameter to the CPN model that can be configured without making changes to the net-structure. Below we provide more details on selected modules of the CPN model. The complete CPN model including all color sets, variable declarations, and function definitions is available from [3].

Fig. 3 shows the client submodule of the Clients substitution transition in Fig. 2. The substitution transition QuorumCalls is used to model the behavior of applications running on the clients, which makes the read and write quorum calls. In particular, the submodules of QuorumCalls serve as test driver modules used to generate system tests for the distributed storage. The content of QuorumCalls

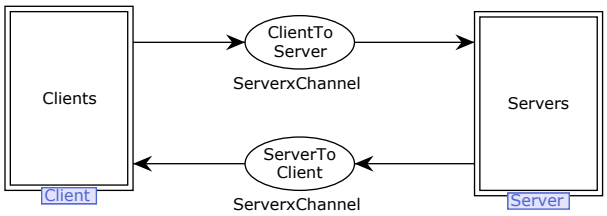


Fig. 2. Top-level module of the CPN model.

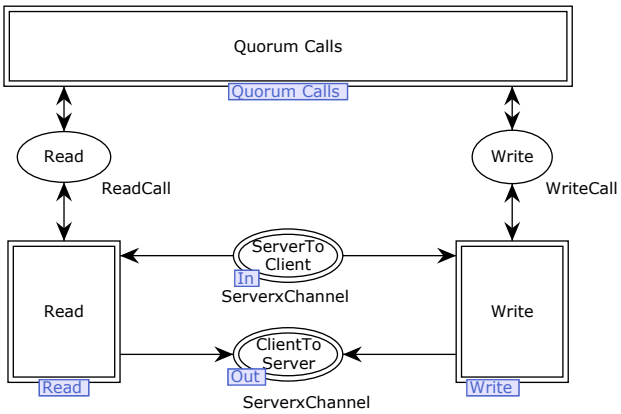


Fig. 3. The Clients module.

depends on the specific test scenarios to be investigated for the system under test, and we give a concrete example of a test driver module in §6. The substitution transitions Read and Write represent the quorum calls provided by the distributed storage. The invocation of quorum calls is done by placing tokens on the Read and Write places. The port places ServerToClient and ClientToServer are linked to the identically named socket places in Fig. 2.

Fig. 4 shows the submodule of the Read substitution transition which provides an abstract implementation of the Read() quorum call. The main purpose of the Read module is to generate test cases for the ReadQF quorum function. A read quorum call is triggered by the presence of a token with the color `READINVOKED(r)`, where `r` identifies the call and is used to match replies from servers to the call. The execution of a read quorum call starts by sending a read request to each of the servers. This is modeled by the transition SendReadReq and the expression on the arc to place ClientToServer, which will add tokens representing read requests being sent to the servers. In addition, a list-token is put on place ReadReplies, which is used to collect the replies received from the servers. The call then enters a WaitingReply state and waits for replies coming back from the servers. When a read's reply comes back, represented by a token on place ServerToClient, then transition ApplyReadQF will be enabled. This transi-

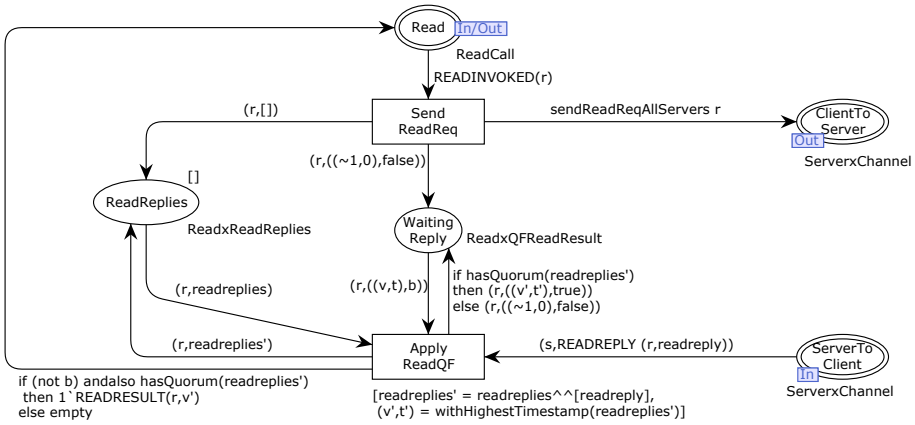


Fig. 4. The Read module.

tion takes the current list of `readreplies` and appends the received `readreply` to form `readreplies'`. The quorum function is then invoked, as represented by the arc expressions to WaitingReply and Read. If enough replies have been received, then a read result is returned to the Read place containing the value with the highest timestamp. As we will see later, we use occurrences of the ApplyReadQF transition for generating test cases for the ReadQF quorum function. In addition, we record the result computed by the CPN model as the test oracle and compare it to the result of our SUT's implementation of the ReadQF quorum function. The submodule for the Write() quorum call is similar. It has a transition ApplyWriteQF, which we use as a basis for generating test cases and obtain a test oracle for the WriteQF quorum function.

Fig. 5 shows the server submodule of the Servers substitution transition in Fig. 2. The replicated state of each server is modeled by the place State. The two substitution transitions are used for modeling the handling of write requests and read requests on the server side. Fig. 6 shows the submodule of the substitution HandleWriteRequest modeling the processing of a write request from a client. The incoming write request will be presented as a token on place ClientToServer and contains a value `v'` to be written in the distributed storage together with a timestamp `t'`. The server compares the timestamp of the incoming write request with the timestamp `t` for the currently stored value `v`. If the timestamp of the incoming write request is larger, then the new value is stored on the server, and a write acknowledgement is sent back in a write reply to the client. Otherwise, the stored value remains unchanged and a negative write acknowledgement is sent to the client in the write reply. The handling of read requests is modeled in a similar manner, except that no comparison is needed, and the server simply returns the currently stored value together with its timestamps.

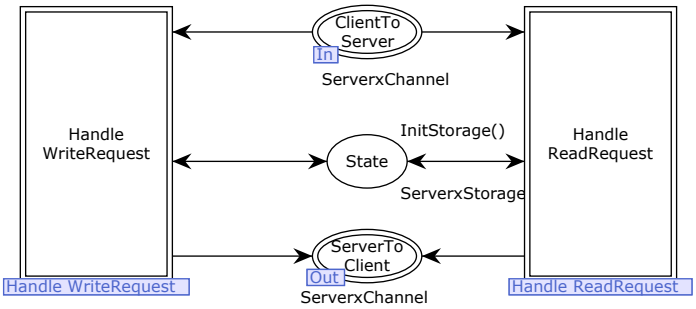


Fig. 5. The Server module.

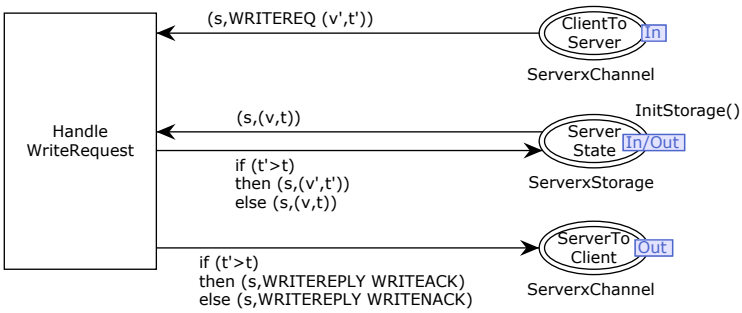


Fig. 6. The HandleWriteRequest module.

5 Test Case Generation

The generation of test cases for Gorums and the distributed storage system is based on the analysis of executions of the CPN model. Test cases can be generated for both the quorum functions and the quorum calls. The test cases generated for the quorum functions are unit tests, whereas the test cases generated for quorum calls are system tests consisting of concurrent and interleaved invocations of read and write quorum calls. The latter tests both the implementation of the quorum calls and the Gorums framework implementation. In addition to the test cases, we also generate a *test oracle* for each test case to determine whether the test has passed.

5.1 Unit Tests for Quorum Functions

Test cases for the ReadQF quorum function can be obtained by considering occurrences of the ApplyReadQF transition (Fig. 4). When this transition occurs, the variable `readreplies'` is bound to the list of all replies that have been received from the servers so far, and which the quorum function is invoked on. In addition, we can use the implementation of the quorum function in the CPN

model as the test oracle. This means that the expected result of invoking the quorum function can be obtained by considering the value of the token put back on place WaitingReply. The value of this token contains the result of invoking the quorum function in its second component. Occurrences of ApplyReadQF can be detected using either state spaces or simulations:

State-space based detection. We explore the full state space of the CPN model searching for arcs corresponding to the ApplyReadQF transition. Whenever an occurrence is encountered we emit a test case together with the expected result. In this case, we obtain test cases for all the possible ways in which the quorum function can be invoked in the CPN model.

Simulation-based detection. We run a simulation of the CPN model and use the monitoring facilities of the CPN Tools [2] simulator to detect occurrences of the ApplyReadQF transition and emit the corresponding test cases. The advantage of this approach over the state-space based approach is scalability, while the disadvantage is potentially reduced test coverage.

Test cases are generated based on detecting transition occurrences. This is done in a uniform way for both detection approaches. Specifically, we rely on a *detection function*, which must evaluate to true whenever a specific transition occurrence is detected. When this happens, a *generator function* is invoked to generate the actual test case.

The generated test cases and the expected results are exported in a custom XML format. As part of future work, we will investigate the use of a general-purpose XML format. Listing 1 shows an example of how our test cases are represented. The test case for the ReadQF quorum function corresponds to two replies (one with value 0 and timestamp 0, and one with value 42 and timestamp 1). With three servers, this constitutes a quorum, and the value returned from the quorum function is therefore expected to be 42 with the timestamp of 1.

```
<Testcase>
   <CaseName>ReadQFTest1</CaseName>
   <Value>
     <ContentTest>
       <ValTest>0</ValTest><TsTest>0</TsTest>
     </ContentTest>
     <ContentTest>
       <ValTest>42</ValTest><TsTest>1</TsTest>
     </ContentTest>
   </Value>
   <ContentExpect>
     <ValExpect>42</ValExpect><TsExpect>1</TsExpect>
   </ContentExpect>
   <QuorumExpect>true</QuorumExpect>
</Testcase>
```

Listing 1. Example of generated test cases for read quorum function.

5.2 System Tests of Quorum Calls

The generation of test cases and expected results is based on the submodule of the QuorumCalls substitution transition (Fig. 3). This module acts as a test driver for the system by specifying scenarios for read and write quorum calls to the underlying quorum system. By varying this module, it is possible to generate different scenarios of read and write quorum calls.

Fig. 7 shows an example of a test driver in which the client executes one read and one write call (concurrent or in any order). Upon completion of these two calls, a new read call is made. The Invoke transitions represent invocations of read and write quorum calls. Each call has a unique identifier (1, 2, and 3) for identifying the call. The write call also has a value (in this case 7) to be written to the distributed storage.

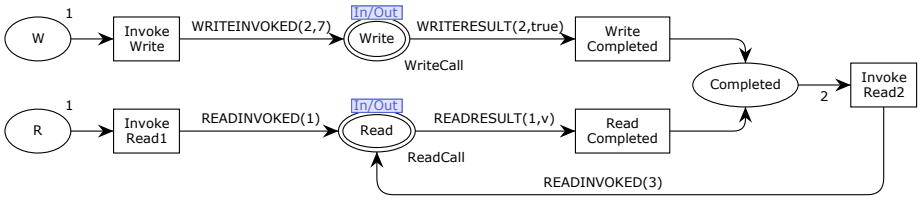


Fig. 7. The QuorumCalls module.

To make test case generation independent to the particular test driver module, we exploit that the read and write quorum calls, made during an execution of the CPN model, can be observed as tokens on the Read and Write socket places (see Fig. 3). When there is a `READINVOKED(i)` token on place READ for some integer `i`, it means that a read quorum call identified by `i` has been invoked. When the read quorum call has terminated, there will be a token with the color `READRESULT(i,v)` present on the place Read, where `v` is the value read by the call. The invocation and termination of write quorum calls can be detected in a similar manner by considering the tokens with the colors `WRITEINVOKED(i,v)` and `WRITERESULT(i,b)` on the place Write (Fig. 3), where the boolean value `b` denotes whether the value `v` was written or not.

Based on this, we can generate test cases in XML format specifying both the concurrent and sequential execution of read and write calls. Listing 2 shows an example where first a read and a write are initiated and upon completion of these two calls, a new read call is initiated. We handle concurrent executions by nesting the read and write `operation` tag as illustrated in Listing 2. In addition, we interpret a new operation positioned after the completion of a call `operation` end tag as the operation should not be started until after the termination of the call. For write calls, we use the value tag to specify the value to be written, and for read calls, we use the value tag to describe the permissible value (see next section) returned by read calls for the test case. The absence of a value between

value tags indicates that the result could be null - corresponding to the case where no value have yet been written into the storage.

```
<RWTest>
  <Function>RWTest</Function>
  <Testcase>
    <CaseName>RWTest1</CaseName>
      <QuoCallMainROpers>
        <OperationName>WRITE</OperationName>
        <OperationValues>
          <Value>7</Value>
        </OperationValues>
          <QuoCallOtherROpers>
            <OperationName>READ</OperationName>
            <OperationValues>
              <Value>7</Value>
              <Value></Value>
            </OperationValues>
          </QuoCallOtherROpers>
      </QuoCallMainROpers>
      <QuoCallMainROpers>
        <OperationName>READ</OperationName>
        <OperationValues>
              <Value>7</Value>
        </OperationValues>
      </QuoCallMainROpers>
  </Testcase>
</RWTest>
```

Listing 2. Example of a generated test cases for the concurrent and sequential execution of read and write calls.

It should be noted if the CPN model specifies that a read and write call may execute concurrently (independently), but happened to be executed in sequence in a concrete execution of the CPN model (e.g., first the read executes and completes and then the write executes an completes), then that will be specified as a sequential test case in the XML format. This is not a problem as the CPN model captures all the possible executions and hence there will be another execution of the CPN model in which the read and the write are running concurrently.

5.3 Test Oracle for System Tests

Checking that the result of an execution with read and write quorum calls is as expected is more complex than for quorum functions. This is because the result of concurrently executing read and write calls will depend on the order in which messages are sent and received. Fig. 8 shows an example test case in which there are two routines (threads of execution) that concurrently execute read and write quorum calls. When $Write_1$ and $Read_a$ are initialized and executed concurrently, the returned result of $Read_a$ could be the old value in the servers before $Write_1$

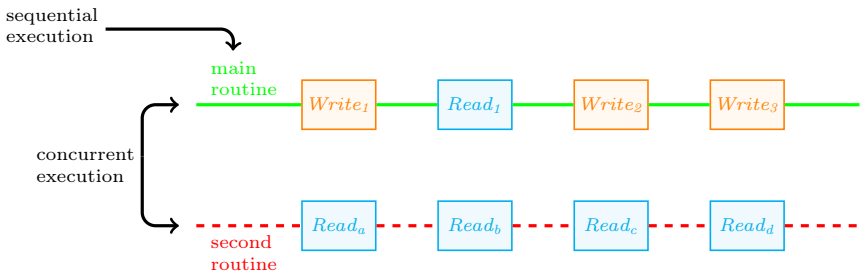


Fig. 8. An example of concurrent and sequential execution of quorum calls.

writes a new value to servers, or the returned result of $Read_a$ could be the value already written by $Write_1$. The same situation applies to $Write_2$ and $Read_c$. Since they are executed concurrently, the returned value of $Read_c$ could be the value written by $Write_1$ or $Write_2$.

This means that if we execute (simulate) the CPN model with a test case containing concurrent read and write quorum calls, then the result returned upon completion of the calls may be different if we execute the same test case against the Go implementation. The reason is that we cannot control in what order the messages are sent and delivered by the underlying gRPC library, i.e., due to non-determinism in the execution. When we apply a state-space based approach for extracting the test cases, e.g., for the quorum function, then we can compute all the possible legal outcomes of a quorum call since the state space captures all interleaved executions. In contrast, we cannot obtain all legal values when extracting test cases from a single execution of the CPN model.

The first step towards constructing a test oracle is to characterize the permissible values of a read quorum call. These are:

1. the initial value of the storage in case no writes were invoked before the read was invoked, or;
2. the value of the most recent write invoked but not terminated prior to the read call (if any) or;
3. the value of the most recent write that has terminated prior to invocation of the read or;
4. the value of a write that was invoked between the invocation and completion of the read.

The above can be formally captured in the stateful automaton shown in Fig. 9 (left), which can be used to monitor the global correctness of the distributed storage. The four events are shorthands for the abstract tokens per client-request observed in the model, e.g., `READINVOKED(i)` is abbreviated RI_i.

The set S is used to collect the set of permissible values for a read call. On a read call RI_i, any pending write $WI(c)$ observed since the last write-return $WI(c)$ is a potential read-result. We abuse notation from alternating automata with parametrized propositions [17] to capture that on a read invocation, we

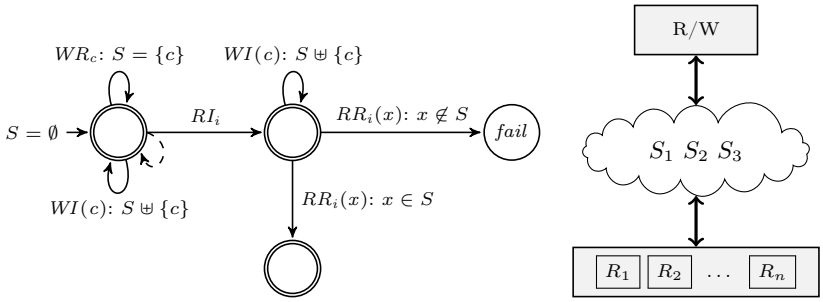


Fig. 9. Read-write automaton (left) and monitor deployment (right).

remain in the initial state and collect further input for a new instance of the monitor with the same current state (indicated by the dashed line) for subsequent read-invocations.

In order to obtain a test oracle which can be used in state space-based and simulation-based test case generation, we use the above automaton to perform run-time verification of the Go implementation when executed on the test cases derived from the CPN model. Specifically, our test adaptor implements a *run-time monitor* corresponding to the above automaton in order to keep track of the invoked and terminated write calls and thereby determining whether a value returned from a read call is permissible. Our test framework currently runs the client (the single writer and multiple readers) within a single Go process. This allows us to directly call into the monitor *before* the client sends the fan-out messages to servers, and *after* the quorum function returns the resulting quorum value, to check the result of the read request for plausibility against the permitted values specified above. This corresponds to monitoring *all* calls and returns in a particular deployment, i.e., correlating read calls and returns of the client in the system against those of the writer in the shaded area of Fig. 9 (right).

6 Testing the Distributed Storage Implementation

We have developed the QuoMBT test framework in order to perform model-based testing of quorum-based systems implemented using Gorums. Fig. 10 gives an overview of the framework which consists of CPN Tools and a test adapter. CPN Tools is used for modeling and generation of test cases and oracles as explained in §4 and §5. The generated test cases are written into XML files by CPN Tools, and then read by the reader of the test adapter, as shown in Fig. 10. The reader feeds the test cases into the distributed storage and each test case is executed with the provided test values as inputs. The tester included in the test adapter compares the test oracle's output against the output of each test case in order to determine whether the test fails or succeeds.

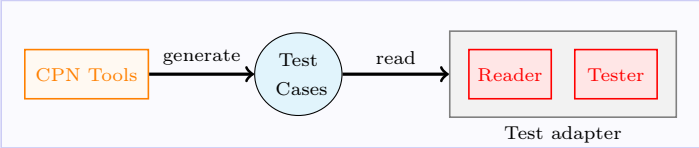


Fig. 10. The QuoMBT test framework.

6.1 The Test Adapter: Reader and Tester

The *Reader* and the *Tester* are both implemented in the Go programming language. The *Reader* can read XML files for unit tests of read and write quorum functions, and for system level tests involving quorum calls. We could have generated Go-based table-driven tests, which is already supported by the Go standard library. However, we chose to use an XML-based format for the generated test cases to enable reuse of the test generator across programming languages.

The implementation of the *Reader* uses Go's *encoding/xml* package, which makes it easy to define mappings between Go structs and XML elements. In order to map XML content into Go structs, each field of the Go struct has an associated XML tag, which is used by Go's XML decoder to identify the field to populate with content from the XML. Also, we have implemented the *Tester* using the *testing* package provided by the Go standard library. Go's testing infrastructure allows us to simply run the `go test` command to execute our generated tests, which will provide pass/fail information for each test case. In addition, this test infrastructure can also provide code coverage.

6.2 Distributed Storage Under Test

To test our distributed storage, we have implemented a test adapter that can execute both the unit tests for user-defined quorum functions and system level tests involving quorum calls. The unit tests for read and write quorum functions can be performed without running any servers, while the system level tests require a set of running servers to test the complete system, including parts of the Gorums framework. When testing the distributed storage, we distinguish between quorum functions and quorum calls, because quorum functions are defined by developers when implementing their specific abstractions, whereas quorum calls are provided generally by the Gorums library. This separation also provides a modular approach to testing.

Our test adapter implements a Go-based *Tester* for testing quorum functions. We simply iterate through the test cases obtained from the *Reader*, invoking the ReadQF and WriteQF functions with the test values, and compare the results against the test oracle. When doing the system level tests involving quorum calls, the servers shown in Fig. 1 must be started first. Then, the test adapter starts a client so that it can execute the quorum calls. The test value, obtained from XML files, for each write quorum call is written to servers by calling the write

quorum call, and for each read quorum call, the value returned by the servers will be captured by the *Tester* to compare against the test oracle. For each write quorum call, the tests only check if it returns an acknowledgment from servers.

The non-trivial part of the test case execution is the concurrent and sequential executions of read and write quorum calls. Fig. 11 illustrates the detailed implementation of the storage involving quorum calls under test. The testing function for quorum calls run through each test case read from the *Reader*. For the run of each test case, the write and read quorum calls can be executed both sequentially and concurrently depending on the test driver used. For the sequential executions, the decision to execute write or read calls is made according to their sequences in the XML files generated by CPN Tools.

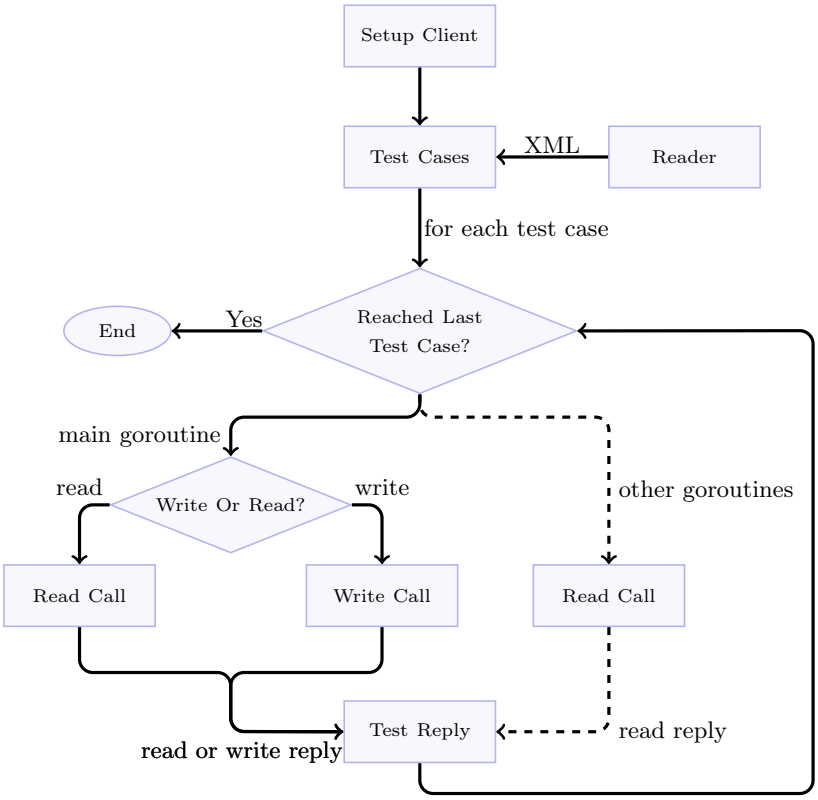


Fig. 11. Flow-chart of test case execution for quorum calls.

For the concurrent executions of write and read quorum calls, the test execution makes use of goroutines provided by the Go programming language. Therefore, in Fig. 11, within each run of test cases, a write or read quorum call (shown by solid line arrows) is executed based on their sequence in the XML files.

Meanwhile, there are other read calls, shown as dashed line arrows in Fig. 11, that can be executed concurrently with the running write or read quorum call shown by solid line arrows. After executing each test case, the returned values of quorum calls are collected.

6.3 Experimental Results

To perform an initial evaluation of our model-based test case generation, we consider the code coverage obtained using different test drivers. The Go toolchain includes a coverage tool which we have used to measure statement coverage.

Table 1 summarizes the experimental results obtained using different test drivers. We consider the following test drivers: one read call (RD), one write call (WR), a read call followed by a write call (RD;WR), a write call followed by a read call (WR;RD), a read and a write call executed concurrently (WR||RD), a read and a write call executed concurrently and followed by a read call ((WR||RD);RD). The table shows the number of nodes/arcs in the state space of the CPN model with the given test driver, the state space and test case generation time (TM) in seconds, the number of test cases generated for quorum calls (QC), the number of test cases generated for quorum functions (QF). For the test case execution, we show the code coverage (in percentage) that was obtained for the system level and unit tests.

Table 1. Experimental results – test case generation and code coverage.

Test Driver	**Test case generation**					**Test case execution** (coverage in percentage)				
						System			**Unit**	
	Nodes	**Arcs**	**TM** (seconds)	**QC**	**QF**	**Gorums Library**	**QCs RD**	**QCs WR**	**QFs RD**	**QFs WR**
RD	39	74	<1	1	3	24.6	84.4	0	100	0
WR	39	74	<1	1	3	24.6	0	84.4	0	100
RD;WR	444	1073	<1	1	7	39.1	84.4	84.4	100	100
WR;RD	615	1376	<1	1	12	40.8	84.4	84.4	100	100
WR||RD	5,119	16,677	6	6	17	40.8	84.4	84.4	100	100
(WR||RD);RD	21,020	59,647	53	6	17	40.8	84.4	84.4	100	100

The results show that the statement coverage for read (RD-QF) and write (WR-QF) quorum functions is 100 % for both system and unit tests, as long as both read and write calls are involved. The statement coverage for read (RD-QC) and write (WR-QC) quorum calls is up to 84.4 %. For the Gorums library as a whole, the statement coverage reaches 40.8 %.

For our system level tests, the statement coverage of quorum calls and Gorums are lower than the coverage for quorum functions. It should be noted that the Gorums library contains all code generated by Gorums' code generator, including gRPC code, various auxiliary functions and logic for the quorum calls.

We have conducted a code inspection, which shows that the statements currently not covered in the Gorums library is code related to error handling. Much of this code is actually not used in case when failures of the system are not considered. Our initial test case generation presented in this paper does not consider failures and error conditions. Hence, with the current testing model, we cannot expect to obtain a higher coverage. The total number of lines of code for the system under test is approximately 2200 lines, which include generated code by Gorums' code generator (around 2000 lines), server code (around 130 lines), client code (around 80 lines) and the code for quorum functions (around 60 lines).

7 Related Work

Research into model-based testing for distributed systems and cloud applications is not new. In Saifan and Dingel's survey [15], they provide a detailed description of how model-based testing is effective in testing different quality attributes of distributed systems, such as security, performance, reliability, and correctness. The authors also classified model-based testing based on different criteria and compared several model-based testing tools for distributed systems according to this classification. This comparison, however did not consider quorum-based distributed systems.

Model-based development and testing for the C♯ language with tool support from Microsoft has been described in [19], and [7] devotes an entire book to describe the same steps that we have taken here in detail: modeling a system as a finite-state machine, and then using state space exploration to derive test cases. In contrast to our work, they do not cover distributed system, but a single application.

Until now, there also has been relatively few applications of CPNs for model-based testing and test cases generation. Watanabe and Kudoh [21] propose two CPN-based test suite generation methods for distributed systems, referred to as the CPN Tree (CPT) method and the CPN Graph (CPG) method. Their method does not directly address a particular way in deriving a CPN model for a distributed system, nor do they give any particular guarantees on achieved coverage for their methods.

Xu suggested using high-level Petri nets for MBT and implemented their approach [22]. The benefits of using high-level Petri nets over finite state machines and UML was: a) the ability to include data in the models and hence directly derive concrete test cases; and b) a compact modeling of parallelism making it simpler to obtain test cases for systems with concurrency. Xu presents the Integration and System Test Automation (ITSA) tool which supports test code generation for languages such as Java, C/C++, and C♯. The ITSA tool also uses the state spaces of the testing model to generate and select test cases. To obtain concrete test cases with input data, the tool relies on a separate model-implementation mapping. In contrast, we obtain the input data for the quorum functions and calls directly from the data contained in the testing model. The ITSA approach also includes test selection techniques and metrics in order to

prune the number of test cases. For testing the distributed storage implementation considered in this paper there was no need for test selection techniques. However, when considering more complex quorum system this will likely be needed also in our approach.

Faria et al. [4] use timed event-driven CPNs to generate test cases for distributed systems. They do not use CPNs as a direct interface to the user, but generate them from UML sequence diagrams. Their tool suite has a slightly different focus, as they instrument a running system to observe the messages specified in the sequence diagrams. They use CPNs for similar reasons as us, namely the large existing body of work on them, and their suitability to model concurrent systems with data encoded in coloured tokens. Their notation of coverage primarily cover the specification, not the code, but recording coverage data on the underlying code should be easy to achieve with conventional means.

A CPN-based test generation approach has been proposed by Liu *et al.* [14]. The approach consists of conformance testing oriented CPN (CT-CPN) as the basic models, a new PN-ioco relation to specify the meaning for an implementation to conform to its specifications, and the test case generation algorithm for simulating the CP-CPN model. For the test case generation algorithm, the authors only considered simulation-based test case generation for the simplified file downloading protocol system. However, in our paper, we also consider state-space based test case generation.

8 Conclusions and Future Work

The main contribution of our work has been to establish an infrastructure consisting of a CPN modeling approach, test case generation algorithms, and a test case execution framework, which can be used to validate quorum-based systems implemented using the Gorums library. Our initial experiments with this infrastructure on a distributed storage system have been promising in that we have obtained a relatively good code coverage even with simple test drivers and a small number of test cases.

An important attribute of our approach is that the CPN testing model has been constructed such that it can serve as a basis for model-based testing of *other* quorum-based systems. In particular, it is only the modeling of the quorum calls on the client and server side that are system dependent. To experiment with different quorum functions for a given quorum system, it is only the implementation of the quorum functions in Standard ML that needs to be changed. The state space and simulation-based test case generation approaches are independent of the particular quorum system under test.

Model-based testing can be used to test a system either by connecting a model (acting as a test driver) directly to an instance of the running system, or, as we do in this paper, generate test cases offline and execute these test cases against the system. The main challenge related to this, is how to handle non-determinism during test case execution. In our current approach, we have addressed this by using monitors known from the field of run-time verification.

The work presented in this paper opens up several directions of future work. We have obtained good coverage results on the quorum functions and calls with the current testing model which encodes a *fair weather scenario*, i.e., it generates test cases where the environment behaves flawlessly by only doing reordering of messages through non-determinism and interleaving in the model. In order to increase coverage of the Gorums library as a whole, we need to test the quorum calls under adverse conditions, such as network errors and server failures. This will require extensions to the model, e.g. injecting erroneous values or generating timeouts. Thus, the recorded test cases must also record the particular scenarios in system tests such that the environment can replay the conditions. Conversely, with the current model, an intermittent failure in the test environment during system testing may be reported as test failures, as they are likely to produce a result diverging from the recorded test output.

Our current solution uses the CPN model to generate test cases and record the correct response from the quorum function. The global monitor presented in §5 independently specifies safe behavior in the form of correct read calls. Instead of the automaton, a different formal specification logic for (distributed) systems could have been used, e.g. Scheffel and Schmitz's distributed temporal logic [16]. Their three-valued logic would allow us to adequately capture that the monitor has neither detected successful nor failed completion.

To evaluate the generality of our modeling and test case generation approach, we need to apply it to other and more complex quorum-based systems. This will challenge the limits of state space-based generation of test cases. It therefore becomes important to investigate the test coverage that can be obtained with simulation versus the test case coverage that can be obtained with state spaces. We anticipate that this will motivate work into techniques for on-the-fly test case generation during state space exploration.

References

1. H. Attiya, A. Bar-Noy, and D. Dolev. Sharing Memory Robustly in Message-passing Systems. *J. ACM*, 42(1):124–142, Jan. 1995.
2. CPN Tools. CPN Tools homepage. http://www.cpntools.org.
3. CPN Testing Model for Gorum-based Distributed Storage. http://home.hib.no/ansatte/lmkr/DistributedStorage.cpn. April, 2017.
4. J. P. Faria and A. C. R. Paiva. A toolset for conformance testing against uml sequence diagrams based on event-driven colored Petri nets. *Intl. J. on Software Tools for Technology Transfer*, 18(3):285–304, 2016.
5. Google Inc. gRPC Remote Procedure Calls. http://www.grpc.io.
6. Google Inc. Protocol Buffers. http://developers.google.com/protocol-buffers.
7. J. Jacky, M. Veanes, C. Campbell, and W. Schulte. *Model-Based Software Testing and Analysis with $C^\sharp$*. Cambridge University Press, 2008.
8. K. Jensen and L. Kristensen. Coloured Petri Nets: A Graphical Language for Modelling and Validation of Concurrent Systems. *Communications of the ACM*, 58(6):61–70, 2015.
9. Jepsen. Distributed Systems Safety Analysis. http://jepsen.io.

10. L. Kristensen and V. Veiset. Transforming CPN Models into Code for TinyOS: A Case Study of the RPL Protocol. In *Proc. of Intl. Conf. on Application and Theory of Petri Nets and Concurrency*, volume 9698 of *LNCS*, pages 135–154, 2016.
11. L. M. Kristensen and K. I. F. Simonsen. *Applications of Coloured Petri Nets for Functional Validation of Protocol Designs*, pages 56–115. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
12. L. Lamport. The Part-time Parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.
13. T. E. Lea, L. Jehl, and H. Meling. Towards New Abstractions for Implementing Quorum-based Systems. In *37th IEEE Intl. Conf. on Distributed Computing Systems (ICDCS)*, 2017. To appear.
14. J. Liu, X. Ye, and J. Li. Colored Petri Nets Model Based Conformance Test Generation. In *IEEE Symp. on Computers and Communications (ISCC)*, pages 967–970. IEEE, 2011.
15. A. Saifan and J. Dingel. Model-based Testing of Distributed Systems. Technical Report 548, School of Computing, Queen's University, Canada, 2008.
16. T. Scheffel and M. Schmitz. Three-valued Asynchronous Distributed Runtime Verification. In *Twelfth ACM/IEEE Intl. Conf. on Formal Methods and Models for Codesign (MEMOCODE)*, pages 52–61. IEEE, 2014.
17. V. Stolz. Temporal Assertions with Parametrized Propositions. *Journal of Logic and Computation*, 20(3):743–757, 2010.
18. M. Utting, A. Pretschner, and B. Legeard. A Taxonomy of Model-based Testing Approaches. *Software Testing, Verification and Reliability*, 22:297–312, 2012.
19. M. Veanes, C. Campbell, W. Grieskamp, W. Schulte, N. Tillmann, and L. Nachmanson. Model-based testing of object-oriented reactive systems with Spec Explorer. In *Formal Methods and Testing*, volume 4949 of *LNCS*, pages 39–76. Springer, 2008.
20. M. Vukolic. *Quorum Systems: With Applications to Storage and Consensus*. Morgan and Claypool, 2012.
21. H. Watanabe and T. Kudoh. Test Suite Generation Methods for Concurrent Systems Based on Coloured Petri Nets. In *Software Engineering Conference*, pages 242–251. IEEE, 1995.
22. D. Xu. A Tool for Automated Test Code Generation from High-level Petri Nets. In *Proc. of ICATPN'2011*, volume 6709 of *LNCS*, pages 308–317. Springer, 2011.

A Tool Chain for Test-driven Development of Reference Net Software Components in the Context of CAPA Agents

Martin Wincierz

Theoretical Foundations of Computer Science (TGI)
Department of Informatics, University of Hamburg, Germany
http://www.informatik.uni-hamburg.de/TGI/

Abstract Testing is common practice in the realm of software engineering. Especially in agile approaches, where test-driven development can be seen as integral.
Capa agents are developed under the agile Paose approach. Their internal components are implemented using Java reference nets which combine the semantics of P/T nets and Java. The existing testing methods for these kinds of nets are either difficult to learn or ill-suited for test-driven development and regression testing.
In this work a tool chain is presented which allows testing of reference nets using regular Java classes. For this purpose an extension of the well-established *JUnit* framework is provided. All tools are designed to be easily understood by developers of Capa agents. This is achieved by a mixture of automatic code generation, repurposing other tools of the Paose approach, and employing a style of testing that is reminiscent of regular Java tests.
While most of the code generating tools are limited to the context of Capa agents, regression testing is possible for any kind of reference net. The findings may help in the development of testing frameworks for other high-level Petri net formalisms.

Keywords: High-level Petri nets, testing, agile development, agile modeling, regression tests, automated tests

1 Introduction

Capa (Concurrent Agent Platform Architecture) allows for the construction of software agents based on reference nets [6]. These are developed under the agile Paose (Petri net-based Agent-Oriented Software Engineering) approach which is described in [4] and expanded upon in [9]. This approach employs the guiding metaphor of the *multi-agent system of developers*. The communicative nature of agile procedures mirror the developed agent systems. Many agile practices such as Pair-programming and common code ownership are already part of Paose. Until now however, there was no explicit support for automatically executed regression tests, let alone test-driven development, both of which are part of many other agile approaches.

The tools presented in this work allow for the test-driven development of regression tests for Capa agent components. These tests are written using the *JUnit* framework which is already well-supported by many continuous integration softwares. Furthermore the techniques presented are at least partly applicable to general reference nets. As such it is of interest to ask if the ability to develop high-level Petri nets under a test-driven approach is useful in a more general case outside of Capa agents. This will be discussed in the first part of this work, before introducing the tool chain in the second.

2 Background

We briefly introduce (Java) reference nets, the general structure of a Capa agent, some important aspects of Paose and the widely used *JUnit* framework.

2.1 Java Reference Nets

Java reference nets, from here on simply referred to as reference nets, were first introduced by Kummer [12]. They support a hierarchical nets-within-net structure (with nets as tokens), as well as Java inscriptions that are executed when a transition is fired [13]. Reference nets can be executed directly by the Renew Petri net simulator with true concurrency semantics. Reference nets in Renew consist of templates and net instances that are generated from these templates. The net instances can be considered as objects. The semantics allow for most Java commands as well as some additional statements like guards and synchronous channels which are explained under Net Interfaces. Tokens are references to reference nets / Java objects. The content of the Java objects can be changed / manipulated by Java code being executed by firing a transition.

Net Interfaces The interfaces of reference nets are implemented with synchronous channels. These consist of two parts: an *uplink* and a *downlink*, as depicted in Figure 1. Downlinks have the form `<target net instance>:<channel name>(<parameter>*)`. Uplinks have the same form, except they do not have a target net instance. When an up- or downlink is enabled, the simulator tries to find a corresponding counterpart with the same channel name and matching parameters. Previously unbound parameters are bound to the value provided by the other channel if applicable. This allows exchange of objects / values / references between net instances. If a matching channel is found and all parameters can be bound to a value, both transitions fire synchronously. More than two transitions can be synchronized by using more channel inscriptions, with the restriction that only one uplink is allowed per transition.

Stub Classes Reference nets are themselves Java objects. To allow easy access to their interfaces one can use *stub classes* which are generated from stub files with Java-like syntax. They basically function as adapters and make net

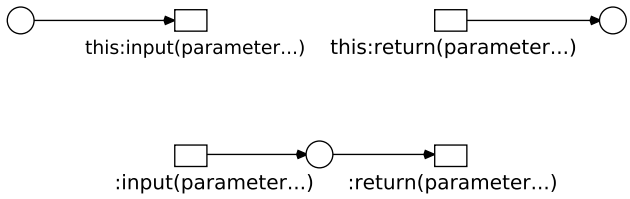


Figure 1. Example of synchronous channels. Top row: Downlinks for the channels named "input" and "return". The target net instance in this case is the same net, represented by the keyword "this". Bottom row: Corresponding uplinks.

instances available for Java classes. Usage of stub classes actually allows to exchange every Java object by a reference net instance and vice versa.

The example shown in Figure 2 shows the stub syntax and resulting Java code for one of the standard interfaces used in PAOSE. The channel name "newExchange" is a standard name that is used multiple times. The specific channel instance is identified with the provided String id. Since this id will never change during runtime, it is declared in the method body. This simplifies the job of testers, as they do not have to know the channel instance of the net. Instead this task is shifted to the person writing the stub file. This is important because the correct implementation of the stub class depends on the use of the interface. Both, uplinks and downlinks, can be used to send and receive information, sometimes both at once. Since Java only allows for a single return value, channels might require multiple corresponding Java methods.

```
1  break void input
2  (Object o,int id) {
3  String s="testChannel";
4  this:newExchange(s,o,id);
5  }
```

```
1  public void input
2  (final Object ppo, final int ppid){
3  ...
4  SimulationThreadPool.getCurrent().
5  execute(new Runnable() {
6    public void run() {
7    de.renew.unify.Tuple inTuple;
8    de.renew.unify.Tuple outTuple;
9    ...
10   outTuple=de.renew.call.
11   SynchronisationRequest.synchronize(
12    _instance,"newExchange",inTuple);
13 }
```

Figure 2. Example for a stub file and generated stub class code. The stub syntax (left) blends elements of Java methods with the syntax of synchronous channels. The resulting Java code can be seen on the right. Synchronization requests are handled by the simulator in the same way as regular transition occurrences are handled. The keyword "break" in the stub file causes the synchronization request to be written in a separate *Runnable*. In this case, it allows us to call the "input"-method several times without having to wait for the simulator to process the request. Later on we use this to emulate a live environment by giving full control to the simulation engine.

2.2 Capa Agents

Figure 3 shows the architecture of a Capa agent. For testing purposes four elements are of interest:

The agent itself is accessible through the "receive" and "send" transitions which are inscribed with synchronous channels of the same name. Messages given as parameters via synchronization are Java objects of a specific type.

Protocols are tied to an act of communication and exist only as long as that communication lasts. They are implemented as instances of reference nets and are held in the place labeled "conversations".

Decision components as well are implemented as instances of reference nets. They are however instantiated when the agent is started and their lifetime is usually the full runtime of the agent itself. They are used to model proactive behavior of the agent, but also to implement services used by multiple protocols.

The exchange transition (located between conversations and decision components) is used to synchronize uplinks of the "dcexchange" channel in protocols with uplinks of the "exchange" channel in decision components. The same channel can be used to allow communication between two decision components (not modeled in the figure). Individual instances of these uplinks are identified via additions to the channel name provided as String parameters. Examples of this can be seen in Figure 4.

Agent-, protocol- and decision component-nets, as well as their respective interfaces, are the main concern when black-box testing Capa agents.

2.3 Some Paose Concepts

Full comprehension of the Paose approach is not required in order to understand this work. Therefore this section will only address two tools which are part of the Paose development process and which are used during testing later on.

Net Components are subnets which serve as templates for recurring functionality within the nets. For example there exist net components for the aforementioned "exchange" channel, as seen in Figure 4. Net Components consist of regular net elements and are only visually distinguishable, i.e. they are easily recognized by humans, but no meta information about them is kept in the net template.

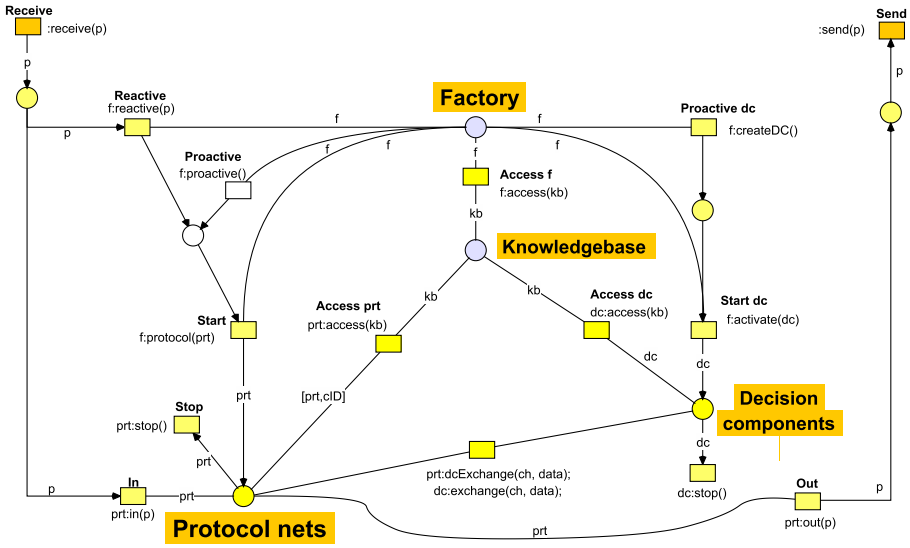


Figure 3. A standard Capa agent, stripped of some elements for better readability.

Agent Interaction Protocols (AIP) are extended UML sequence diagrams. They are used to model interactions between different agents[1]. An example can be seen later in Figure 7. It is possible to automatically generate net skeletons of protocol nets from the models. This is done by mapping the inscriptions on the diagram elements to Net Components which are then drawn and connected in the same order.

2.4 The JUnit Framework

JUnit is a testing framework for Java applications [1]. It is designed for the creation of regression tests. Automatic execution of *JUnit* tests is supported by many continuous integration programs. The testing process is shown in Figure 5.

Tests can be started, manually or automatically, from the IDE or from the command line using build tools. To execute the tests a controller class called test runner is used. Often test classes specify which runner is supposed to execute them.

The runner creates a `TestResult` object which is usually used to generate a test report. The design of the report depends on the implementation but often includes the number of failed and succeeding tests, the expended time and the stack trace in case of a failure.

JUnit4 heavily relies on reflectively manipulating tests by use of annotations. The `@RunWith` annotation is used to specify the runner class. `@Test` marks the tests themselves. Optionally a time limit may be set, which is useful if deadlocks

[1] More specifically the interactions between agent roles.

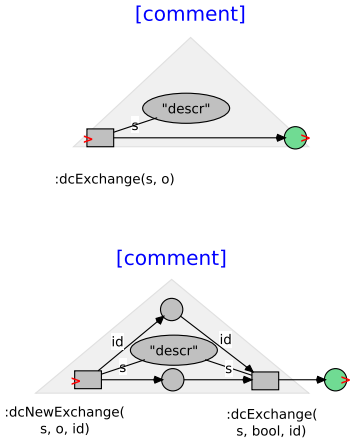


Figure 4. Net Components to be used in decision component nets. The placeholder String "descr" is replaced by the channel name identifying the channel instance and is given as the parameter 's'. The other parameters are 'o', an object reference which is either given or received, and 'id', an integer id created by the agent to map requests to answers.

are a concern. `@Before` and `@After` are called before and, respectively, after each test. They are used for setting up the testing environment and returning it to its prior state after testing.

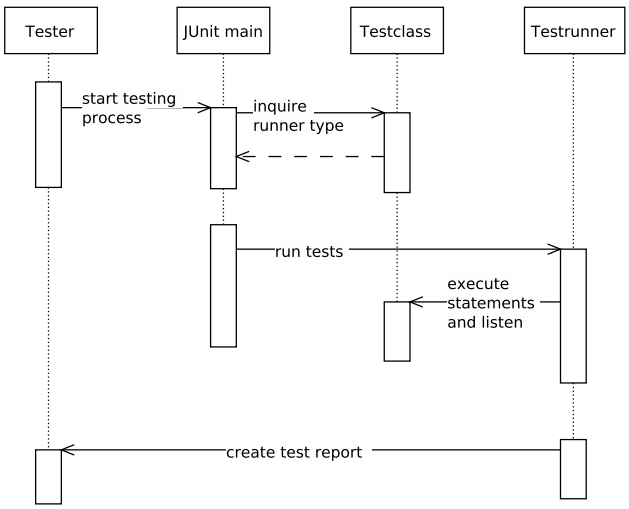


Figure 5. A sample interaction diagram for the *JUnit* testing process.

3 Petri Nets in Agile Development

The core practices of Agile Modeling are introduced and it is shown that Petri nets can be used in conformance with them. It is also argued why Petri nets are especially useful as a modeling language in agile development.

3.1 Agile Modeling

Agile Modeling, as it is presented by Ambler [2], is, much like agile development, not a specified process, but attempts to be a guideline to modelers. "Agile Modeling is not a prescriptive process. [...] it does not define detailed procedures for how to create a given type of model, instead it provides advice for how to be effective as a modeler." [2] Its ideas mirror those of agile development and Ambler explicitly shows its conformity with *eXtreme Programming* [2]. *Agile Modeling* is based on its own set of principles which is put into action via eleven core practices organized into four categories [2]. In this section it is shown that Petri nets can be used according to these practices and are therefore applicable to *Agile Modeling*.

Iterative and Incremental Modeling The first practice is *apply the right artifacts*. Petri nets cannot be considered a universal modeling language but are useful in modeling concurrent behavior. For these kinds of tasks, they are indeed the right artifacts. Similarly the practice *iterate to another artifact* is easy to fulfill if we assume that Petri nets are not our only means of modeling. If one is stuck during the modeling of a net, switching to a different modeling task may bring clarification.

The practices *create several models in parallel* and *model in small increments* require a specific style of nets. More precisely they require nets that are limited in their scope. This can be achieved by hierarchical net structures, as is done in Renew with the nets-within-net approach or CPNTools with hierarchical Coloured Petri Nets [11]. The nets-within-net formalism even allows the exchange of subnets at runtime.

Teamwork The practices of this category are *model with others*, *active stakeholder participation*, *collective ownership* and *display models publicly*. All of this is part of the Paose approach and has been successfully done within the context of a Petri nets-based software development environment. [9]

Simplicity This category encompasses the practices *create simple content*, *depict models simply* and *use the simplest tools*. Again, it is assumed that Petri nets are only used to model concurrent software components. The nets can be refined from simple P/T nets into high-level Petri nets. There exists a number of modeling tools but for early designs they can also be easily drawn by hand.

Validation The first practice of this category, *consider testability*, is the main subject of this work. The second, *prove it with code*, is elaborated on in Section 3.2.

3.2 Combining Model and Implementation

It has been have established that Petri nets can be used as a modeling language in agile development. To take this a step further it is shown that Petri nets are especially useful in agile approaches due to the nets being executable.

Research suggests that there are significant advantages keeping models and code consistent [8]. This is, however, difficult to achieve in agile projects, as the software is continually evolving. Reference nets are Turing equivalent and used as both a modeling and implementation language in our Paose approach. This ensures that the model automatically evolves alongside the code, as they are indeed the same. Petri nets are therefore highly suitable for agile development processes.

The design / testing paradigm favored in *eXtreme-Programming* and other agile approaches is test-driven development [3]. Tests are written before the implementing code and serve as a guideline to programmers. Instead of being seen as additional work after the actual programming job is done, testing is part of the design itself [7]. Executable Petri nets can be seen as both, model and implementation. If they are used in agile development, it is only natural to design them according to agile practices. Therefore the approach to test Petri nets always also incorporates the notion of test-driven modeling.

4 Related Work

The idea of test-driven modeling has been proposed before.

Hawari et al. [10] used the phrase of *test-driven* models. They did not include the technical framework, but rather followed the idea of testing for different parameter simulations. In the following we illustrate how to set up the modeling approach for the use of current software engineering methods.

Zhang and Patel have successfully used similar techniques with executable UML in industry software projects [19,20]. Their process is very close to the one presented later. "First, we create the UML sequence diagrams, then we create both UML model and test cases (for unit, integration, and system testing) according to the sequence diagrams." [19] This work, however, is the first to apply this to Petri nets.

Walkinshaw and Derrick [17] follow the idea of inferring (automata) models from Erlang code and to generate model-based tests according to possible traces of the models to test software. They herewith avoid the involvement of users and gain an automated test generation. In the approach presented the models can be used directly and therefore do not need to be guessed or deduced from some code executions. This is one of the advantage when following a model driven

software engineering approach where the models can directly be used for code execution as in PAOSE .

In the realm of Petri nets, testing is more associated with the techniques of verification or model checking. The presented approach is not competitive, but complementary to these. If the state-space becomes too large or the problem simply becomes undecidable due to added semantics, testing might be a good alternative.

5 Requirements

Decisions that have been made regarding some aspects of the tools are clarified. Requirements and motivations that guided the development are explained.

5.1 Functional and Non-functional Requirements

The testing methods presented are specifically designed to satisfy the needs of agile development. For this purpose three main points that had to be incorporated were identified:

Test-driven Development Adapted to Petri nets, this means tests can be written even before the net structure is known. In the field of testing this is known as black-box testing [15]. Rather than a finished program only the interfaces of the (net-)object are needed. Tests are designed to fail and the code is written iteratively to gradually fulfill the testing requirements.

Small-scale Unit- and System-wide Integration Tests Kent Beck recommended when talking about *eXtreme Programming*:
"If the gap [between writing code and tests] is minutes, the cost of communicating expectations between two people would be prohibitive." [3] The short iterations require the availability of small-scale unit-tests. These are tests of a single net or a small grouping of nets. Beck also acknowledges that usually these tests are not enough:
"A programmer or even a pair bring to their code and tests a singular point of view [...]." He therefore proposes: "One set [of tests] is written from the perspective of the programmers, [...] another set is written from the perspective of customers or users [...]." [3] These tests use the interfaces open to customers. They are system-wide tests, that can be used to avoid unexpected side-effects when integrating smaller software-modules into larger systems. In the context of RENEW and its reference net formalism, which was the main concern of our testing efforts, there is no need to differentiate between the views on a technical level. The nets-within-net property of reference nets allow for hierarchical structures within the nets. System tests are therefore unit-tests of nets that are high in the hierarchy. Trivially, all of these tests have to be regression tests. Once written, they can be called multiple times. During the implementation phase

this is usually done manually by programmers in order to guide them in writing code. Once a test successfully completes, it is called automatically every time new code is integrated into the software. This is done to avoid unexpected side effects and is known as integration testing [15].

Usability and Heterogeneous Skill Sets The importance of this last point is difficult to quantify for a more general case. In academic software-projects however, a significant discrepancy in the abilities of programmers has been observed. As a joint Bachelor's and Master's project, both seasoned programmers with several years of working experience, as well as beginners who have never used a UNIX operating system before are working together [14]. The upfront workload and difficulty of learning PAOSE techniques for the first time often proved to be a hurdle. Therefore one of the goals for this work was to create a testing framework and tool chain that is as easy to use as the rest of the PAOSE techniques, but is as self-explanatory as possible to not further increase the learning time.

5.2 Choosing a Test Language

To write tests the testing-framework *JUnit* is used. Its use was already suggested in earlier works, however favoring a hybrid solution that relied on *JUnit* only for starting and controlling tests. In the first approach the tests themselves were written in the language of the test-objects, i.e. with reference nets [5,16]. In this contribution it was decided on a different approach. The tests are fully written as Java classes. In [18] the approach has been implemented and tested. There are several reasons for this choice.

Familiarity As mentioned earlier, some of the participants of academic CAPA software projects have never seen Petri nets before, let alone used them for programming. However in order to even create something that is in need of testing, a certain degree of Java knowledge can be assumed. By relying solely on Java, tests are not much different from what a Java programmer would usually write, therefore shortening the time needed to learn the testing process. For developers that are completely new to programming with nets, we can also assume that the tests themselves are much less prone to failures and errors compared to the new language of Petri nets. Especially in the context of test-driven development, in which tests are written before the implementation, it is likely that the first attempts using nets are faulty, effectively defeating the purpose of writing these tests.

Support Features RENEW provides some support features to developers, but not nearly as many as comparable IDEs for wide-spread high-level programming languages. The main draw of programming with reference nets, the concurrency, is completely irrelevant for the tests themselves.

JUnit Functionality *JUnit* provides additional functionality, for example methods that are called before each individual test or expecting a test to fail due to an exception.

Separation of Test and Implementation The implementation of the tested functionality is completely separated from its test. The net instance used can be exchanged for a different kind or even a Java class. Apart from technical advantages, this also better conforms to the goals of testdriven development. Theoretically, tests created this way could be written without any knowledge of Petri nets, preventing any chance of the tests being influenced by the implementation.

6 The MulanNetTest Plugin

This Renew plugin was developed as part of a Bachelor's thesis by the author. It was later expanded upon by adding support for the *JUnit* framework. All tools will be explained using the "WebChat" example. A use case can be seen in Figure 6.

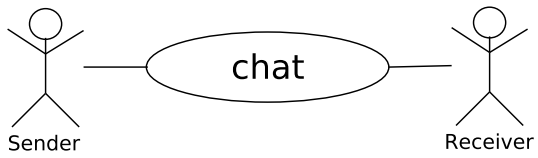


Figure 6. Use case of a simple web chat. A chat message is sent from a 'sender' agent to a 'receiver' agent.

6.1 Extending the AIP

The first task when writing tests before the implementation is done, is to provide interfaces against which can be tested. For agents and protocols this is easy. Agents only provide their standard interfaces. Protocols can be generated from the AIP. Decision components however are entirely created by the developers.

To solve this, the AIP were extended to include both types of internal components. An example for the WebChat application can be seen in Figure 7. Depending on the outgoing and incoming arrows, as well as their inscriptions, net components are generated and connected in the order according to the diagram. From the model in Figure 7 four nets are generated: Two protocol nets and two decision component nets. The net 'Sender_DC' can be seen in Figure 8. The commentary fields employ the new comment tool. It allows developers to append blue text to net elements which is also added to the elements' meta information and can later be retrieved.

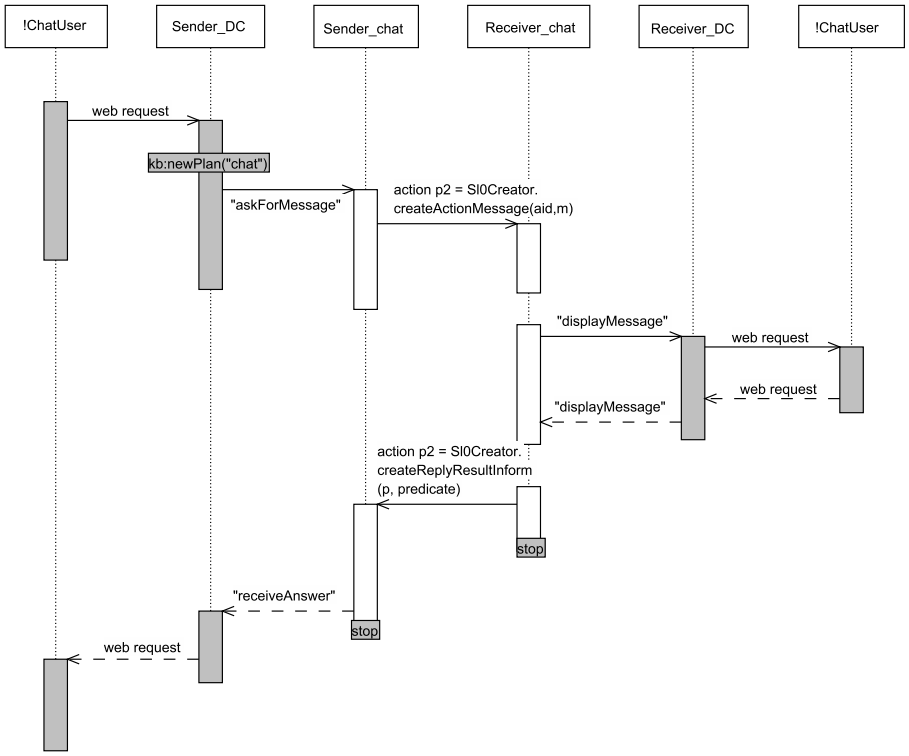


Figure 7. The white figures in the middle are associated with protocols and are part of the original AIP. The gray figures represent decision components. The figures marked with an '!' do not generate net skeletons and represent external access, in this case through a standardized CAPA web interface.

6.2 Automatic Net Stub Generation

In order to test the newly generated nets with *JUnit*, an adapter stub class has to be created. This is done automatically by mapping the standardized Net Component interfaces to code in net stub syntax. The Components are identified by naming the place containing the channel name after the interface type. For all places that have non-standardized names getter/setter methods are created. The result can be seen in Figure 9. The commentary is read from the place's meta information and is also written into the final Java class. The stub generation is done within the IDE via a context menu, as seen in Figure 10.

6.3 A JUnit Adapter for Petri Nets

To write the tests themselves the *JUnit* framework is employed and an adapter for this was created. First all newly added elements are explained. An in depth example is given in the next section.

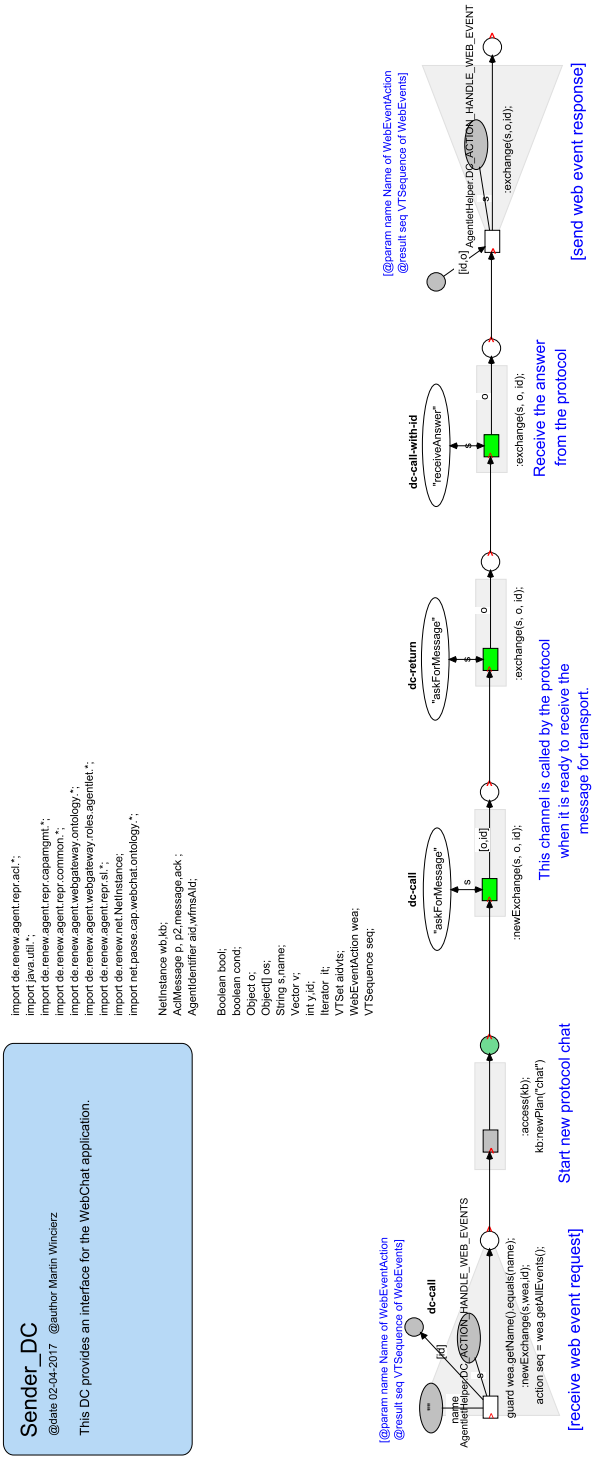


Figure 8. One of the nets generated from the AIP. The elements were rearranged to improve readability and the comment fields and channel names were filled in. Otherwise no new elements were added.

```
1        /**
2        Receive the answer
3        from the protocol
4        */
5        void receiveAnswer(Object o, int id){
6
7        String s="receiveAnswer";
8        this:exchange(s,o,id);
9        }
```

Figure 9. One of the stub methods generated from the net skeleton. The method names are derived from the channel names. If multiple methods share the same name, they are numbered. Illegal characters are automatically removed.

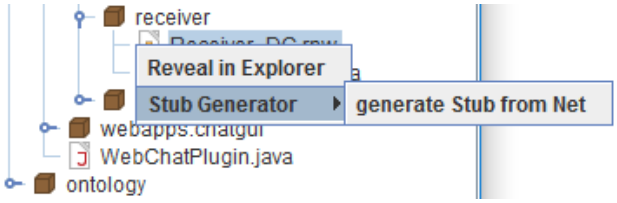


Figure 10. The context menu to generate net stubs. The stub is created in the same folder as the target net and carries the same name supplemented by the suffix "Stub".

Annotations The adapter mirrors *JUnit4*'s use of reflection. New annotations are `@Repeat(int)`, which allows easy repetition of a test class, and `@Concurrent Parameters(Object[][])`, which is modeled after the `@Parameters` class used with the `Parameterized` test runner. It is used to define an array of arrays. For each entry of the super-array, a new test instance will be created and run concurrently. The sub-array values are reflectively injected into fields annotated with `@ConcurrentParameter(int)` according to the given adicity. This is also modeled after the regular *JUnit* functionality of the `Parameterized` runner. Thus users who have used it before, will hopefully understand the principle immediately.

RenewTestRunner The adapter is designed to be very close to regular *JUnit* in its use, as to allow easy adoption of its functionality. The core element is a custom test runner, which automatically starts a new Renew instance. The runner is designed to support the new annotations.

RenewTestClass All tests have to extend this class. This is more akin to the older *JUnit* version 3, where all tests had to extend a `TestClass`. It is necessary because of Renew 's plugin based architecture. New instances of Renew are run in a new class loader to ensure the correct order of loading the plugins. The test runner cannot read the objects annotated with `@ConcurrentParameters`, therefore this task as well as the field injection is done by the test class itself upon

being called reflectively. In addition the `RenewTestClass` provides convenience methods to synchronize testing phases. This is elaborated on in the example.

6.4 Example Test Class

The code example shows a simple test class. The artifact to be tested is the "Sender_DC" net. During setup a new instance of the net is created (line 17/18), which is then wrapped in a stub adapter at the beginning of the test (line 24). The net instance is static because it will be used by the three instances of the test class with the different specified parameters.

The tests are synchronized using the `finishPhase()` method (lines 23,34,38). The test will wait until all instances of the test class have finished the current phase. The phases can be distinguished as executing and evaluation phases. Between each phase the simulation engine is halted / restarted. This ensures that during execution all test instances are active in the same part of the net, thus possibly provoking concurrency failures if existent.

Not all stub methods have been shown, but the methods can be easily matched to their respective channels in the net graphic by their names. In this case only the first half of the net is tested. A message is received from the web interface and given to the protocol net. The test concludes successfully, if both messages are the same. Since no functionality has been implemented, the test will fail after 3000 milliseconds.

```
1  @Repeat(times = 3)
2  @RunWith(value = RenewTestRunner.class)
3  public class WebChatTest extends RenewTestClass {
4
5    @ConcurrentParameters
6    public Object[][] params = {{"house", 1}, {"car", 2}, {"petri", 3}};
7
8    @ConcurrentParameter(value = 0)
9    public String message;
10
11   @ConcurrentParameter(value = 1)
12   public int id;
13
14   static NetInstance instance;
15
16   @Before
17   public void setup() {
18     Net net = Net.forName("Sender_DC");
19     instance = net.buildInstance();
20   }
21
22   @Test(timeout = 3000)
23   public void testMessaging() {
24     this.finishPhase();
25     Sender_DCStub stub = new Sender_DCStub(instance);
26     WebEventAction wea = new WebEventAction();
27     WebEvent we = new WebEvent();
28     VTSequence vts = new VTSequence();
29
30     we.setData(this.message);
31     vts.add(we);
32     wea.setEvents(vts);
33     wea.setName("");
```

```
34
35        stub.AGENTLET_DC_ACTION_HANDLE_WEB_EVENTSIn(wea, this.id);
36        this.finishPhase();
37
38        stub.askForMessageIn(Boolean.TRUE, this.id);
39        String result = stub.askForMessageOut(this.id);
40        this.finishPhase();
41
42        Assert.assertEquals(this.message, result);
43      }
44
45    }
```

7 Discussion

The code generation tools are useful in the context of Capa agents, but difficult to extend to more general cases. The unification mechanism of synchronous channels makes a mapping to Java methods problematic, as the number of required methods increases exponentially with the number of parameters, since any combination of receiving and giving Objects has to be taken into account. Other high-level Petri net formalisms with more specified interfaces might be more suitable.

The *JUnit* extension however allows the testing of all reference nets, provided a net stub is manually created. Writing the tests is about as difficult as testing regular Java classes and should not require special knowledge about Petri nets.

The testing methods have proven to be able to find semantical failures in Capa applications. Other properties, like liveness, are not possible to determine using testing. For this a combined approach with verification techniques might be a solution.

Also the *JUnit* extension in its current form is costly to use. To provide a clean environment, a new instance of Renew is started before each test. This takes about three to five seconds depending on the hardware on which the tests are run.

Despite some flaws the tools presented allow for easier testing and thereby higher quality of code. Automatic test execution will likely improve the integration process during development, although this has to be further observed in practice. The increased degree of testability improves the usefulness of reference nets not only as a programming, but also as a modeling language.

8 Conclusion

High-level Petri nets can combine graphical feedback with early simulation and might therefore be suitable as a modeling language in agile development. Especially of interest is the notion of test-driven modeling, which has been done before with other modeling languages, but has not been tried with Petri nets. A tool chain was presented with which test-driven development of Capa agents becomes possible. The agent's internal components are reference nets which function as both implementation as well as their own model. The testing process was shown using a simple example.

References

1. Junit4. `http://junit.org/junit4/`. Accessed: 2017-01-05.
2. S. Ambler. *Agile Modeling: Effective Practices for EXtreme Programming and the Unified Process.* Programming, software development. Wiley, 2002.
3. Kent Beck. *Extreme Programming Explained.* Addison Wesley, Boston, 2005.
4. Lawrence Cabac. *Modeling Petri Net-Based Multi-Agent Applications*, volume 5 of *Agent Technology – Theory and Applications.* Logos Verlag, Berlin, 2010. http://www.sub.uni-hamburg.de/opus/volltexte/2010/4666/.
5. Lawrence Cabac, Michael Duvigneau, Daniel Moldt, and Matthias Wester-Ebbinghaus. Towards unit testing for Java reference nets. In Robin Bergenthum and Jörg Desel, editors, *Algorithmen und Werkzeuge für Petrinetze. 18. Workshop AWPN 2011, Hagen, September 2011. Tagungsband*, pages 1–6, 2011.
6. Michael Duvigneau, Daniel Moldt, and Heiko Rölke. Concurrent architecture for a multi-agent platform. In Fausto Giunchiglia, James Odell, and Gerhard Weiß, editors, *Agent-Oriented Software Engineering III. Third International Workshop, Agent-oriented Software Engineering (AOSE) 2002, Bologna, Italy, July 2002. Revised Papers and Invited Contributions*, volume 2585 of *Lecture Notes in Computer Science*, pages 59–72, Berlin Heidelberg New York, 2003. Springer-Verlag.
7. Steve Freeman and Nat Pryce. *Growing Object-Oriented Software, Guided by Tests.* Addison Wesley, Upper Saddle River, NJ, 2010.
8. Andrew M Gravell, Yvonne Howard, Juan-Carlos Augusto, Carla Ferreira, and Stefan Gruner. Concurrent development of model and implementation. 16th International Conference on Software & Systems Engineering and their Applications, event date: 2/12/2003, 2003.
9. Matthias Güttler. Integration einer agilen Projektmanagementumgebung in ein verteiltes Team. Diploma thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg, November 2013.
10. Aliah Hazmah Hawari and Zeti-Azura Mohamed-Hussein. Simulation of a Petri net-based model of the terpenoid biosynthesis pathway. *BMC Bioinformatics*, 11(1):83, 2010.
11. Kurt Jensen and Lars M. Kristensen. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems.* Springer Publishing Company, Incorporated, 1st edition, 2009.
12. Olaf Kummer. *Referenznetze.* Logos Verlag, Berlin, 2002.
13. Olaf Kummer, Frank Wienberg, Michael Duvigneau, Lawrence Cabac, Michael Haustermann, and David Mosteller. Renew – the Reference Net Workshop. Available at: `http://www.renew.de/`, June 2016. Release 2.5.
14. Dennis Schmitz. Neugestaltung von Lernmaterialien zur Unterstützung der Lehrenden und Lernenden in der petrinetzbasierten und agentenorientierten Softwareentwicklung. Master thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg, February 2016.
15. Uwe Vigenschow. *Testen von Software und Embedded Systems: professionelles Vorgehen mit modellbasierten und objektorientierten Ansätzen.* dpunkt, Heidelberg, 2010.
16. Florian von Stosch. Entwicklung eines Testrahmenwerks für Mulan-Protokollnetze. Bachelor thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg, September 2012.
17. Neil Walkinshaw and John Derrick. Incrementally discovering testable specifications from program executions. In Frank S. de Boer, Marcello M. Bonsangue,

Stefan Hallerstede, and Michael Leuschel, editors, *Formal Methods for Components and Objects - 8th International Symposium, FMCO 2009, Eindhoven, The Netherlands, November 4-6, 2009. Revised Selected Papers*, volume 6286 of *Lecture Notes in Computer Science*, pages 272–289. Springer, 2009.

18. Martin Wincierz. Erweiterung des PAOSE Softwareentwicklungsansatzes um ein Testkonzept und Bereitstellung von Plugins zu dessen technischer Umsetzung. Bachelor thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg, 2016.
19. Y. Zhang and S. Patel. Agile model-driven development in practice. *IEEE Software*, 28(2):84–91, March 2011.
20. Yuefeng Zhang. Test-driven modeling for model-driven development. *IEEE Software*, 21(5):80–86, Sept 2004.

Part III

Short Papers

Modeling Reusable Concurrent Passive Entity Objects in Colored Petri Nets

Rowland Pitts and Hassan Gomaa

George Mason University, Fairfax, Virginia, USA
{rpitts,hgomaa}@gmu.edu

Abstract. Concurrent software systems are growing increasingly large and complex; the risks associated with poor design and architectural choices are increasing as well. Building executable prototypes can help identify problems early and Colored Petri Nets are well suited to this purpose. This paper presents an approach to modeling reusable thread-safe passive entity objects in Colored Petri Nets, including public, private and static members, plus encapsulation and object composition.

Keywords: colored Petri nets, concurrency, rapid prototyping, passive entity object.

1 Introduction

Concurrent software systems are growing increasingly large and complex. Consequently, the risks associated with poor design and architectural choices are increasing as well. Assembling executable models can help to identify problems early, and Colored Petri Nets (CPN) [9] are well suited for building executable concurrent software models; additionally, the language primitives facilitate the modeling of reusable design pattern templates [7], as well as the passive entity objects they interact with.

In spite of the fact that failure is increasingly expensive [1], often little consideration is given to system performance or reliability until a project is already implemented; unplanned behavioral analysis is typically inefficient, unreliable and difficult to repeat [12].

CPNs routinely depict concurrent software systems as tokens moving through a series of operations (transitions), sequentially or navigating control structures, analogous to dynamic flow charts. This paper introduces an approach to modeling thread-safe objects, with an emphasis on object-oriented properties, such as information hiding, providing a public interface of operations, and reusability using CPN Tools [5].

This paper is organized as follows: Section 2 discusses related work, Section 3 introduces the modeling approach and Section 4 provides validation. Section 5 discusses conclusions and future work.

2 Related Work

There is much literature devoted to the analysis of concurrent software with CPNs, and some related to object modeling.

Bauskar and Mikolajczak modeled objects using CPN's hierarchical capabilities [3]. Jensen and Kristensen have examined reusability using CPNs hierarchical capabilities [9]. Costa and Gomes propose module replication, composition and defining interfaces [4]. Barros and Gomes discuss transitions as functions with input parameters and also the creation and destruction of objects [2]. Pettit, Fant and Gomaa have modeled behavioral design patterns and communication templates, including threads-of-control [7, 12, 11]. Lakos introduces Object Petri Nets, which incorporate inheritance, polymorphism, dynamic binding, and include a single class hierarchy of both token and subnet types [10]. The Reference Net Workshop supports object references as tokens [13].

This paper focuses on combining a number of object-oriented properties while modeling concurrent objects, such as information hiding, providing a public interface of operations, static variables and operations, and reusability, as well as modeling threads-of-control by which a client can animate passive entity objects as needed.

3 Object Modeling in CPN

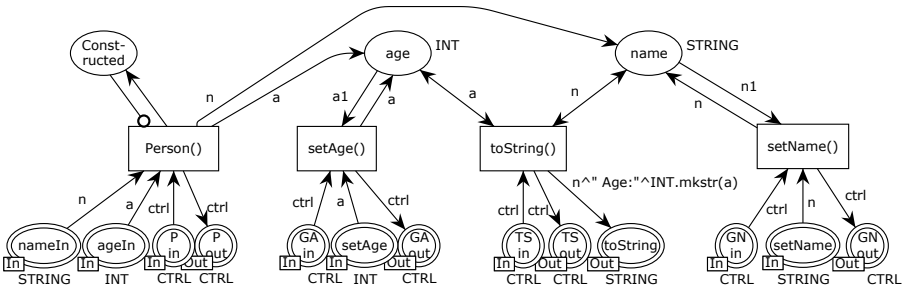


Fig. 1. Person class definition in CPN.

3.1 Design Conventions

CPNs are not inherently object-oriented; however, the language primitives allow for almost infinite flexibility. To the extent that visual structure aids in conveying a designer's intent, the following conventions, illustrated in Figure 1, are utilized for object modeling. The behavior otherwise modeled in Figure 1 is discussed in more detail in the next subsection.

Input and Output Parameters are depicted across the bottom of their class diagrams, grouped by operation, and indicated by a double-line, as opposed to a single line. This includes threads-of-control, which determine the sequence in which modeled operations execute. Placing tokens into the input places, and retrieving tokens from the output places, is the means by which clients communicate with objects.

Operations comprising a class' public interface are represented as transitions just above, and connected by arcs to, their respective inputs and outputs. Modeling operations as transitions works well for multiple reasons: transitions perform conversions, and CPN Tools' hierarchical capabilities facilitate the creation of reusable objects that effectively enforce communication through the defined public interface and otherwise prevent access to an object's non-public members.

Instance Variables are depicted as places in the space above the public interface operations, or as objects as described herein, and are maintained by the public operations or by other internal functions.

3.2 A Simple Class Example

Figure 1 is the class definition for a simple Person class. Two places near the top represent instance variables for age and name. Four public operations are provided: `Person()`, `setAge()`, `toString()` and `setName()`, and each is represented by a transition. Their various inputs and outputs are represented by places across the bottom.

Concurrency: No two operations can simultaneously access an object's values. No operation can execute until the constructor has been initially executed. Furthermore, the constructor cannot re-execute after the object is created.

Encapsulation: When used, a Person object's data elements and functionality are encapsulated within the object, providing the client with only indirect access through the defined public operations. An example Employee object is depicted in the uppermost region of Figure 2.

Reusability: Any number of Person objects may be used within a CPN.

3.3 A More Complex Example

Figure 2 represents an Employee class definition, which features a composed object (`Person1`), a static variable (`employeeCount`) and associated static accessor method (`getCount()`), and a meta-variable (`lock`) used for synchronization. Employee also includes a constructor (`Employee()`) and a `toString()` operation.

Given the relative complexity of this example, representing an operation with a single transition is insufficient. Treating these as atomic actions would result in the thread-of-control being released to the client prematurely, and potentially cause concurrency issues. Therefore, an inbound transition fires to initiate the behavior sequence, and a return transition fires when the process is complete, releasing the thread-of-control and return values at the appropriate time.

For simplicity, a minimal number of operations have been modeled; however, more could easily be added. For example, `setAge()` and `setName()` could be added, and connected to the otherwise unused equivalents in `Person1`.

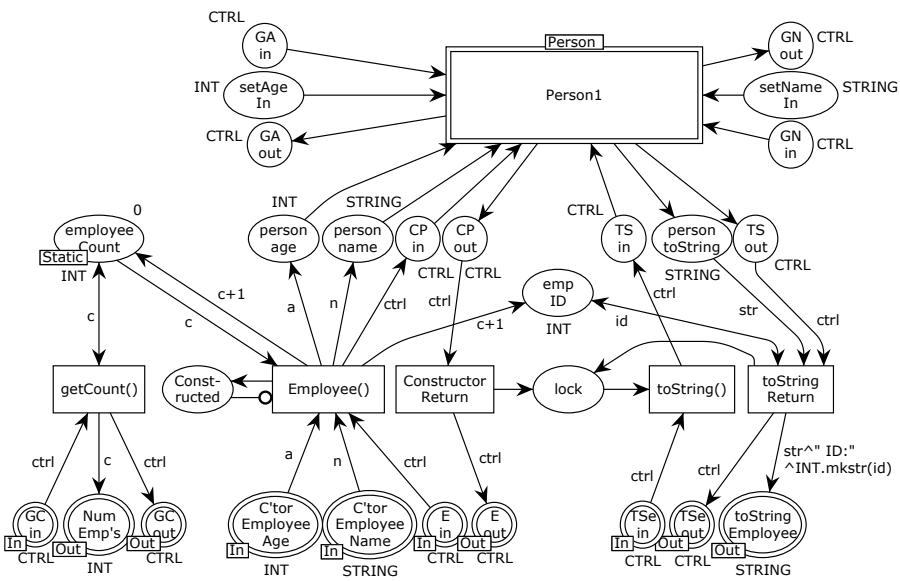


Fig. 2. Employee class definition, with composed Person object and static members.

Concurrency: The Employee class employs a more explicit locking mechanism. When the constructor executes, a token is moved to the `lock` place. To ensure mutually exclusive access, each instance-method must acquire the lock before executing and return it when finished [8]. Therefore, no two can execute simultaneously (given the scope of this short paper, only one such method is depicted, but any additional methods would acquire and release the lock token in the same way). Non-static methods cannot execute until the constructor has been invoked to create the object, and the constructor cannot re-execute once the object is created.

Encapsulation: An Employee object's data elements, including a Person object, and functionality are encapsulated. Employee provides only indirect access to itself through the defined public operations.

Reusability: Any number of Employee objects may be used within a CPN; additionally, each Employee object also re-uses a Person object.

Static Behavior: The `employeeCount` place is effectively made static by defining it as a fusion place, facilitated by CPN Tools. Its initial marking is zero (simulating an initialized value), and is incremented each time an instance of Employee's constructor is invoked. The value in the fusion place is shared by all instances of Employee; therefore, invoking the `getCount()` method in any instance of Employee will return the same value.

4 Validation

The limited length of this short paper permits only a brief description of the validation carried out; however, tests were conducted to determine that each modeled object's operations execute correctly and that the synchronization considerations ensure that there is no detrimental conflict for shared data. A unit testing approach was employed, because it offers "the most effective means to test individual software components for boundary value behavior" [6]. Figure 3 is a depiction of one such test scenario

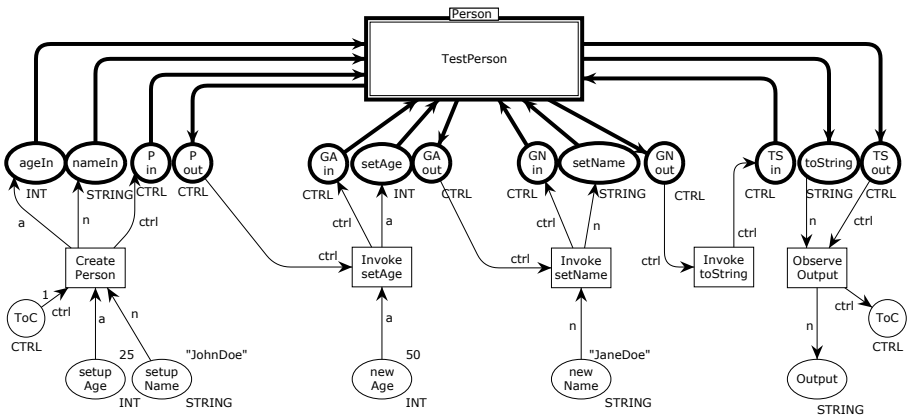


Fig. 3. Unit test of a Person object.

For clarity, the object under test, including the input and output places associated with its public operations, is depicted with bolder lines. The elements otherwise associated with the testing operations are depicted normally.

Test Scenario: From left to right in Figure 3, a Person object is created, after which the `setAge()` and `setName()` operations are invoked. Finally, the `toString()` operation is invoked in order to observe the expected output.

5 Conclusions and Future Work

Objects can be effectively modeled with CPNs, as shown in the examples above. The unit tests conducted confirm that they perform as expected. Modeling single- and multithreaded active objects is the logical next direction related to this short paper. Modeling inheritance would pose an interesting challenge as well.

This work is part of a larger project to model concurrent distributed applications and middleware. The ultimate goal of the overarching research effort is to provide a suite of executable architectural components and communications templates for a variety of software design patterns.

References

1. Ammann, P., Offutt, J.: Introduction to Software Testing. Cambridge University Press (2016)
2. Barros, J.P., Gomes, L.: On the Use of Coloured Petri Nets for Object-Oriented Design, pp. 117–136. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
3. Bauskar, B.E., Mikolajczak, B.: Abstract node method for integration of object oriented design with colored Petri nets. In: Third International Conference on Information Technology: New Generations (ITNG'06). pp. 680–687 (April 2006)
4. Costa, A., Gomes, L.: Module composition within Petri nets model-based development. In: 2007 International Symposium on Industrial Embedded Systems. pp. 316–319 (July 2007)
5. CPN Tools website (May 2017), `http://cpntools.org`
6. Ellims, M., Bridges, J., Ince, D.C.: Unit testing in practice. In: 15th International Symposium on Software Reliability Engineering. pp. 3–13 (Nov 2004)
7. Fant, J.S., Gomaa, H., Pettit, R.G.: A comparison of executable model based approaches for embedded systems. In: 2012 Second International Workshop on Software Engineering for Embedded Systems (SEES). pp. 16–22 (June 2012)
8. Gomaa, H.: Real-Time Software Design for Embedded Systems. Cambridge University Press (2016)
9. Jensen, K., Kristensen, L.M.: Colored Petri nets: A graphical language for formal modeling and validation of concurrent systems. Commun. ACM 58(6), 61–70 (May 2015)
10. Lakos, C.: Object Oriented Modelling with Object Petri Nets, pp. 1–37. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
11. Pettit, R.G., Gomaa, H., Fant, J.S.: Modeling and prototyping of concurrent software architectural designs with colored Petri nets. In: International Workshop on Petri Nets and Software Engineering. pp. 67–79 (2009)
12. Pettit, R.G., Gomaa, H.: Modeling behavioral design patterns of concurrent objects. In: Proceedings of the 28th International Conference on Software Engineering. pp. 202–211. ICSE '06, ACM, New York, NY, USA (2006)
13. The reference net workshop website (May 2017), `http://www.renew.de`

Part IV

Poster Presentation

Towards a Systematic Model-driven Approach for the Detection of Web Threats and Use Cases

Simona Bernardi[1], Raúl Piracés Alastuey[2], Alejandro Solanas Bonilla[2], and Raquel Trillo-Lado[2]

[1] Centro Universitario de la Defensa, Zaragoza, Spain, simonab@unizar.es
[2] Universidad de Zaragoza, Spain, raul.piraces@gmail.com, 647647@unizar.es, raqueltl@unizar.es

Abstract. The increasing use of Web Information System has made them an attractive target for attackers. Herein, we present firsts results and current work on a method for improving the security of such systems, which is based on Model-Driven Engineering and Process Mining.

Introduction. The Web has become a popular communication and information exchange channel, not only for people but also for different types of systems. Thus, for example, while previously cyber physical systems, such as the electrical networks, were isolated; nowadays, they are usually interconnected via information infrastructures where the Web is used. For example, the company Iberdrola Distribución Eléctrica offers its clients a service to consult their electrical consumptions via Web applications[3]. The increasing use of Web Information System has made them an attractive target for attackers. According to the last Symantec report published in April 2017 [2] *"Web attacks are still a big problem, with an average of more than 229,000 being detected every single day in 2016"*. Besides, the same report indicates that *"More than three-quarters (76 percent) of scanned websites in 2016 contained vulnerabilities, nine percent of which were deemed critical"*. So, improving the security of Web Information Systems in order to detect new threats and vulnerabilities is relevant, in particular in the context of critical infrastructures such as energy networks.

Approach overview. Recently, we have proposed a new method based on Model-Driven Engineering and Process Mining techniques for improving the security of Web Information Systems [1]. Our proposal consists of five main steps:

- *Step 1.* Specification of the expected system behavior by means of the Unified Modeling Language (UML) [3].
- *Step 2.* Automatic generation of a Petri net model from the UML-based specification by means of the DICE-tools [4]. This model formally specifies the expected system behavior and it is named *normative model*.
- *Step 3.* Control and monitoring of the Web Information System to get data logs that are evidences of the operative (or real) behavior of the system.

[3] https://www.iberdroladistribucionelectrica.com/consumidores/inicio.html

- *Step 4.* Pre-processing of the data logs to transform them into *event logs* for process mining.
- *Step 5.* Use of process mining techniques for the identification of deviations between the normative model and the operative behavior. The deviations are analyzed to determine if they are potential threats or new trends of use (new use cases) or missed use cases not considered when the normative model was specified.

When a potential threat is detected, new measurements to mitigate or remove the risk of its materialization are considered and deployed. On the other hand, when a new use case is detected it is analyzed to improve the services provided to the users (e.g., to offer customized services to the clients or to improve the usability of the Web system). Missed use cases are used to enhance the initial UML specifications and improve the performance of the method proposed.

The method was used to study the SID Digital Library[4] by considering its logs during the last seven years. Very promising results, that demonstrate the feasibility of the proposal, were achieved: new trends of usage were identified and threats, previously not detected, were discovered [1].

On-going work. To improve the approach, we are currently tackling several open issues such as: 1) define new heuristics to get event logs that enable the analysis on different levels of granularity; 2) develop plugins to automatize the method and enable the analysis of logs on-line; 3) create a library of attack patterns for testing purposes; and 4) apply the method to new case studies[5].

Acknowledgment. This work has been funded by the projects: "Desarrollo de técnicas de detección de ciberataques en sistemas de información mediante minería de procesos" [UZ-CUD2016-TEC-06], "Ciber-resilient critical infrastructures: Exploiting process mining techniques for security-by-design" [CyCriSec-TIN2014-58457-R], and "Developing Data-Intensive Cloud Applications with Iterative Quality Enhancements" [DICE-H2020-644869].

References

1. S. Bernardi, R. Piracés-Alastuey, R. Trillo-Lado, Using Process Mining and Model-driven Engineering to Enhance Security of Web Information Systems, 2nd Int. Workshop on Safety & Security aSSurance for Critical Infrastructures Protection (S4CIP), 29th April, 2017, Paris (France).
2. Symantec Corp. Global Internet Security Threat Report, vol. 22, April 2017.
3. Object Management Group. Unified Modeling Language (UML), v2.5, June 2015.
4. A. Gómez, C. Joubert, J. Merseguer, A Tool for Assessing Performance Requirements of Data-Intensive Applications, pp. 159–169, XXIV National Conference of Concurrency and Distributed Systems, 2016, ISBN: 978-84-16478-90-3.

[4] SID Digital Library: `http://sid.cps.unizar.es/BiD`

[5] The social network *Yarning*: `https://www.yarning.es` and the Content Management System *e-ditor*: `http://www.e-ditor.es`.

Towards Verification of Connection-Aware Transaction Models for Mobile Applications

Lars M. Kristensen[1] and Gabriele Taentzer[2] and Steffen Vaupel[2]

[1] Western Norway University of Applied Sciences
lmkr@hvl.no
[2] Phillips-Universität Marburg
{taentzer,svaupel}@informatik.uni-marburg.de

Abstract. Applications running on mobile devices are subject to frequent changes in connectivity to back-end infrastructure. In order not to disrupt service and ensure fault-tolerant operation, transaction-oriented mobile applications must be able to operate in both online and offline mode. Recently, a generic software architecture has been proposed [4] to accommodate mobile transaction models that support offline transaction processing in conjunction with data replication, reintegration, and synchronisation. We present an initial Coloured Petri Net (CPN) [2] model of a mobile transaction system and report on the first results on verifying its behavioural correctness using model checking.

Introduction. Mobile client applications often need to execute transactions that read and write shared data sets stored on a server-side infrastructure. Examples include applications involving local payment, where concurrently running applications need access to funds from a shared account. A challenge in this scenario is that mobile devices may often loose connectivity. To avoid disruption of service, the application must be able to operate even when the mobile device is *offline*. This requires specialised transaction models that replicate data for offline operation and which synchronise data when coming back *online*.

Several conflict-free transaction models have been proposed to support such scenarios. As an example, the Escrow transaction model [3] is based on a logical split of the shared data set, and can be used to for instance give a mobile application access to a restricted amount of funds on an account. Vaupel et al. [4] have proposed a generic architecture that includes online and offline transaction processing, replication, synchronisation and re-integration of data and which is able accommodate different conflict-free mobile transaction models.

CPN model. Our goal is to develop a formal executable specification of the mobile transaction architecture proposed in [4]. In particular, we want to verify the correctness of conflict-free transactions for a given mobile application. Furthermore, the CPN model should reflect the architecture and make it easy to change the set of transactions for a concrete mobile application.

Figure 1 shows the CPN module of the local transaction manager on the mobile client for an example with a Debit transaction operating on a shared

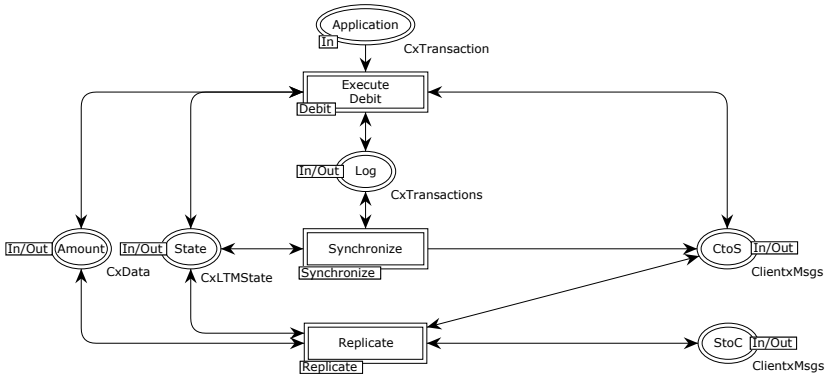


Fig. 1. Local Transaction Manager module.

Amount. The application invokes the debit transaction via the Application place. When operating in offline mode, the transactions executed are written in a Log. The substitution transitions Synchronise and Replicate represent the two major operational modes that allow data to be synchronised with the server-side when online, and conflict-free replication of data to support offline operation. The places CtoS and StoC are used for modelling the communication between the client-side and the server-side.

Verification. We perform verification using explicit-state model checking, as supported by CPN Tools [1]. The state space for the Escrow-based payment transaction system with a debit transaction has $7,174$ states and $22,202$ edges and can be generated in less than three seconds. The transaction model replicates the amount on the account such that all mobile clients have access to an equal amount. A key property of the application is that independently of how the clients go online and offline, it should always be possible to return to a *consistent state* in which the sum of the amounts replicated to the clients is equal to the total amount stored on the server-side. In computation tree logic (CTL), this property can be expressed as $AG\,EF\,p$, where p is a state predicate expressing that the state is consistent with respect to the amount.

References

1. CPN Tools home page. `www.cpntools.org`.
2. K. Jensen and L.M. Kristensen. Coloured Petri Nets: A Graphical Language for Modelling and Validation of Concurrent Systems. *Communications of the ACM*, 58(6):61–70, 2015.
3. P. O'Neil. The Escrow Transactional Method. *ACM Transactions on Database Systems*, 11(4):405–430, 1986.
4. S. Vaupel, D. Wlochowitz, and G. Taentzer. A Generic Architecture Supporting Context-Aware Data and Transaction Management for Mobile Applications. In *Prof. of MobileSoft'16*, pages 111–122. ACM, 2016.

Petri Net with RFID Distributed Database for Autonomous Search and Rescue in Trails and Crossings

João Paulo da Silva Fonseca[1,2] and José Jean-Paul Zanlucchi de Souza Tavares[2]

[1] Universidade Federal de Goiás, Goiânia, Brazil
jpsfonseca@ufg.br
[2] Universidade Federal de Uberlândia, Uberlândia, Brazil
{jpsfonseca,jean.tavares}@ufu.br

Abstract. A modified Petri Net inside RFID database is proposed to assist search and rescue in trails and crossings. The main idea is presented directing rescue agents and trekkers in external areas without the guarantee of satellite communication and with restriction of points with electric power.

Keywords: Search and Rescue · Multiagent systems · PNRD

1 Introduction

According to [1], from 1998 to 2011 the Rocky Mountain Rescue Group assisted 1857 search and rescue (SAR) incidents involving 2198 victims in the USA. From these, 345 were climbing incidents with 428 victims. According to the Brazilian Fire Department, from 2013 to 2014 the occurrences of people lost in forests increases 17,30% in the State of São Paulo and 21.42% in the Baixada Santista [2].

In this way, the SAR community is open to new methodological proposals. Specifically talking, several studies report the use of mobile robotics and Petri nets to aid in the modeling and analysis of SAR operations. For example, a technique to test robot behavior in an urban SAR environment uses elementary Petri nets to model the behavior of system actors [3], and a qualitative analysis of process of triage in disaster rescue operations using stochastic Petri nets [4].

This ongoing work is based on an approach called Petri Net inside RFID Database (PNRD) [5], and proposes to trace the user in external environment with a modified PNRD, with that, to direct the robots' local search in case of any incident.

2 Preliminary Discussion

SAR agents compose the search and rescue team. There is an AerialBot, which is a cartesian robot that can be on Patrol mode, Aid mode or Available mode. There

are two GroundBots, ground mobile robots which can be on Available mode, Local search mode, Aid mode, Approach mode, or Rescue mode. All the SAR agents have a Wi-Fi module, enabling TCP/IP communication. The Walkers, which are the trekkers, are represented by mobile robots that can assume three invariant states: the walker status, the healthy status, and the trek status. The walker status can be on Lost mode, On route mode, or Rescued mode; the healthy status can be on Healthy mode, Injured Mode, or On triage mode; and the trek status can be on Ready to trek mode Ongoing mode, Returning mode, or Full trail mode. Milestones with RFID tags are used at specific points along the trail to facilitate the users' traceability and direct a possible search for injured or missing users.

The RFID readers are embedded on Walker and GroundBots with the modified PNRD Engine, and can read the existing tags along the trail. The equipment records the users' data in the tags, transforming them into a local database of users who traveled there. This equipment will also be able to identify the user's trajectory by storing the possible routes through an incidence matrix. Thus, the PNRD approach is modified so that the reader stores the process (incidence matrix) and trek status (region that the user is in), and the tag starts to store the trigger vector.

This ongoing work presents a new approach to SAR systems without satellite communication using autonomous robots and modified PNRD. The PNRD approach is modified to meet SAR system requirements, storing the incidence matrix and the user's trek status in the RFID reader and using the RFID tag as a repository of the trigger vector and the list of users that have passed through it. The trail was tagged at several points to assist the users' tracking. The models need to be embedded and the tests be performed so the proposal is better discussed. Also, it is intended to expand it to cover colored, timed, hierarchical, and stochastic Petri nets.

References

1. Lack, D.A., Sheets, A.L., Entin, J.M., Christenson, D.C.: Rock climbing rescues: causes, in-juries, and trends in boulder county, Colorado. Wilderness & Environmental Medicine 23(3), 223-230 (2012). doi: 10.1016/j.wem.2012.04.002
2. Rossi, M.: Número de pessoas perdidas em trilhas aumenta 20% no litoral de SP [Online], G1 Santos (2015). Available: http://glo.bo/1CuKTjU [Accessed April. 2017].
3. Andrews, A., Abdelgawad, M., Gario, A.: World model for testing urban search and rescue (USAR) robots using Petri nets. In: Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development, pp. 663-670. Rome (2016). doi: 10.5220/0005782106630670
4. Jianjie, L., Zhaohui, H., Xuan, Y., Ran, Z., Chengan, X., Yuan, L.: Analysis of process of triage in disaster rescue action using stochastic Petri net. In: Proceedings of the 2012 International Conference on Industrial Control and Electronics Engineering, pp. 111-115. Xi'an (2012). doi: 10.1109/ICICEE.2012.38
5. Tavares, J.J.P.Z.S., Saraiva, T.A.: Elementary Petri nets inside RFID database (PNRD). In-ternational Journal of Production Research 48(9), 2563-2582 (2010). doi: 10.1080/00207540903564934

Part V

MoSEBIn Paper

Modeling Mobile Agents in Vehicular Networks

Oscar Urra and Sergio Ilarri

Department of Computer Science and Systems Engineering
University of Zaragoza, I3A, Spain
ourra@itainnova.es, silarri@unizar.es

Abstract. Vehicular networks (VANETs) are mobile ad hoc networks where vehicles that are near each other can exchange data by using wireless communications. Advances in mobile communication technologies have spurred significant research in the exploitation of these types of networks to develop different kinds of data services for drivers. However, these networks are also highly-dynamic and several data management challenges arise to realize their full potential.
We have proposed the use of mobile agent technology for efficient distributed query processing in VANETs. Mobile agents are software entities with the capability to hop among nearby cars in such a way that they can transport themselves to the vehicles storing relevant data. In this paper, using Petri nets, we present a formal model of our approach.

Keywords: Mobile agents, vehicular networks, query processing, data management, Petri nets

1 Introduction

Vehicular networks (VANETs) [3] are mobile ad hoc networks dynamically established among vehicles by using short-range wireless communications and based on communication standards such as WAVE (IEEE 802.11p) [6]. Using these networks, vehicles can exchange relevant data for drivers, such as information about accidents or obstacles on the roads, traffic conditions, available parking spaces, or other moving entities of potential interest. However, several data management challenges appear to fully exploit the potential of VANETs [1]. Most difficulties are due to the fact that the nodes in the network (i.e., the vehicles) are continuously moving, which renders the communication links quite volatile (nodes may appear and disappear at any time and the communication between two distant vehicles is only possible by using multi-hop routing protocols). Thus, for example, two vehicles moving in opposite directions in a highway at high speeds will be within the communication range of each other only during a quite small time window, which constrains the amount of data that can be exchanged.

On the other hand, mobile agents [2, 4] are software entities that have the capability to autonomously move from one computer/device to another during their execution. This technology can bring interesting benefits in distributed systems. Rather than transferring large amounts of data to a node for processing,

the code can be moved to the node storing the data for local processing and filtering, thus saving significant network resources. We have previously proposed the use of mobile agents for data management in VANETs [5]. With our approach, a mobile agent can flexibly take autonomous decisions and jump from car to car as needed to reach the target area and query the data sources within that area.

2 Overview of the Data Retrieval Approach

We consider the problem where we need to retrieve data about a certain spatial area, called *interest area* (IA) or target area. We assume that some vehicles within that area can provide the required data. For example, imagine that some vehicles are equipped with sensors of different types (pollution sensors, noise sensors, videocameras, etc.) and our goal is to retrieve environment data about a geographic area, for monitoring or surveillance purposes. In these circumstances, we could try to flexibly exploit the required sensors available in vehicles traveling through those areas. Similarly, we could want to process a query using a distributed approach that access several vehicles in an area to retrieve data from their local databases, such as data exchanged with neighbor vehicles about available parking spaces nearby. Figure 2 shows an overview of the process:

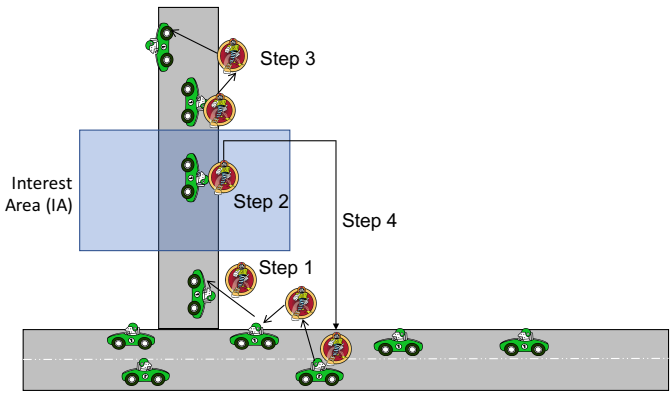


Fig. 1. Overview of the process.

- Step 1: the agent needs to reach the *interest area* (IA).
- Step 2: the agent retrieves data from the vehicles inside the area.
- Step 3: if the vehicle carrying the agent leaves the interest area, the agent will need to find an appropriate strategy to come back.
- Step 4: once the agent has finished the monitoring process, it will need to come back to its origin device and return the results collected.

2.1 Step 1: Traveling to the Interest Area

First of all, the mobile agent needs to reach the interest area. It should be noted that the agent can travel from one place to another by using two complementary mechanisms: by hopping among vehicles (transportation using wireless communications) and by staying in a moving vehicle (transportation via locomotion, using the cars "as taxis").

Figure 2 shows a Petri net that models this stage of the process. The mobile agent is initially created in a given vehicle, as represented by the initial mark in the place "In_Vehicle". Whereas the agent has not succeeded in its attempt to reach the interest area (condition "IA_not_Reached"), it evaluates if there is another vehicle within the communication range that could be a better candidate to transport it to the area. If a better candidate is found (condition "Better_Found"), the agent jumps there, and otherwise it stays in the same vehicle. The process continues until the agent reaches the target area ("IA_Reached"). Notice that there is a transition ("Vehicle_approaching") injecting marks into

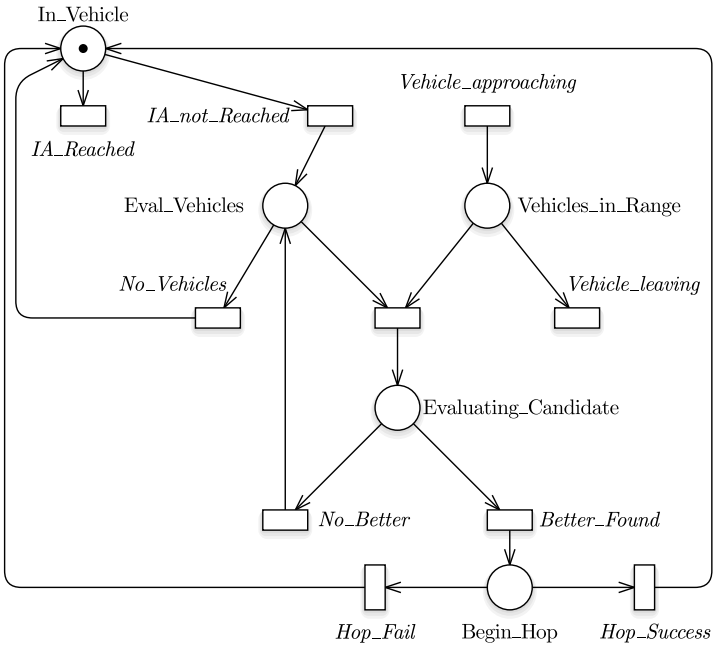


Fig. 2. Model of an agent traveling to the target area.

the place "Vehicles_in_Range" and another transition removing marks from that place ("Vehicle_leaving"), representing the fact that, at any time, new vehicles can start being within (and out of, respectively) the communication range of the vehicle currently transporting the agent. It should be noted that the agent

could apply a variety of strategies to decide if a vehicle is a better candidate or not; for example, a simple greedy approach could select as a better candidate any vehicle which is closer to the target area than the current vehicle.

2.2 Step 2: Monitoring the Interest Area

Once the agent is within the interest area, it has to retrieve data by traversing the area and retrieving data from the vehicles located within the area. The agent considers that spatial area as divided into a certain number of spatial cells (of the same size). We assume that the agent needs to visit at least N cells to finish its task (alternatively, we could require visiting a minimum number of cars). Notice that, by increasing or decreasing the number of cells and the value of the parameter N, we could achieve a more fine-grained or coarse-grained monitoring.

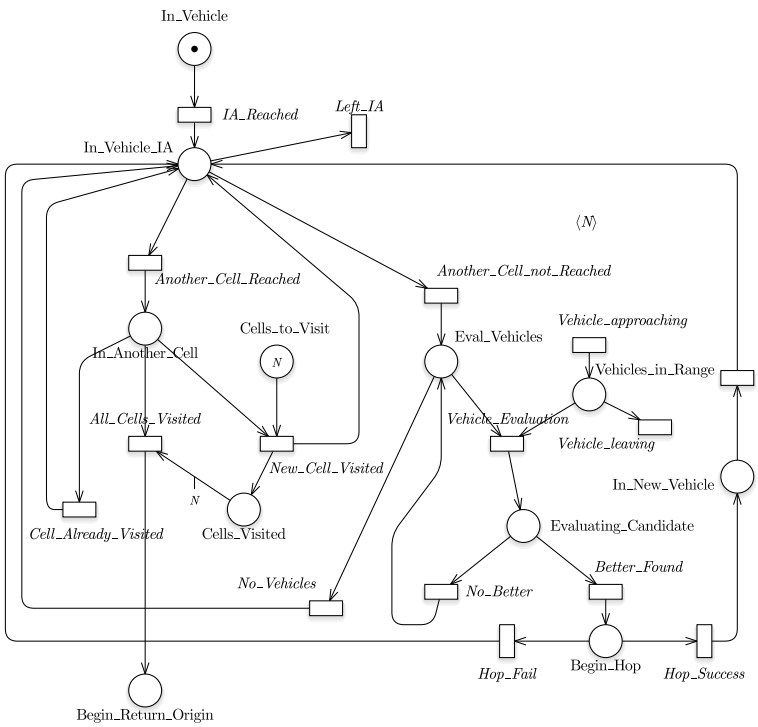


Fig. 3. Visiting the cells within the area: detailed view of the process.

Figure 3 shows a Petri net that models the process of visiting the required spatial cells within the area. Each time that there is an opportunity to visit a cell that has not been previously visited, the agent tries to visit it and, if it succeeds, a mark is injected into the place "Cells_Visited"; if not (tran-

sition "Another_Cell_not_Reached"), it will try to reach the cell by traveling to other vehicles if necessary. Once the cell has been visited (transition "Cell_Already_Visited"), the agent will try to visit a different cell (the mark representing the agent returns to "In_Vehicle_IA" and the agent will consider a different cell). When there are no marks left in the place "Cells_to_Visit", which means that there are N marks in the place "Cells_Visited", the transition "All_Cells_Visited" is fired and the agent finishes this stage of the process (a mark representing the agent is put in the place "Begin_Return_Origin") to start the last stage (step 4).

2.3 Step 3: Returning to the Target Area, if needed

Notice that, as shown in Figure 3, the mobile agent can leave the target area if the vehicle that carries it leaves the area (transition "Left_IA"). In that case, the agent will need to temporarily interrupt the monitoring process and find a way to come back, again by jumping from car to car as needed.

Figure 4 shows the process followed by the mobile agent when the car that carries it leaves the interest area.

2.4 Step 4: Returning to the Origin

In the last phase of the process, the mobile agent needs to return to the device/computer that created the agent in the first place, which may be a moving vehicle. For that purpose, we assume that there is an estimation of the location of that device, in such a way that the mobile agent can take jumping decisions to try to reach that location; in case the location is imprecise, the agent will of course need to expand its searching focus to try to reach its "home device". Due to space constraints, we omit the Petri net due to its similarity with Figure 2.

3 Prospective Work

Some advanced aspects of our proposal are not modeled in the Petri nets shown in this paper, such as the potential use of clones (an agent that creates copies of itself) in situations where the reliability or performance of the data retrieval process may be in danger (e.g., scenarios with a low density of vehicles or where the vehicles follow trajectories that do not pass near the target area). Another interesting aspect to consider is the possibility to annotate the Petri nets with performance metrics and probabilities, in order to exploit them for performance evaluation; an important difficulty to achieve this is that we would need to quantify first the impact that different elements in a scenario (density of vehicles, their trajectories, etc.) can have on the performance of the process.

Acknowledgments

This work has been supported by the projects TIN2016-78011-C4-3-R (AEI/FEDER, UE), TIN2013-46238-C4-4-R, and DGA-FSE.

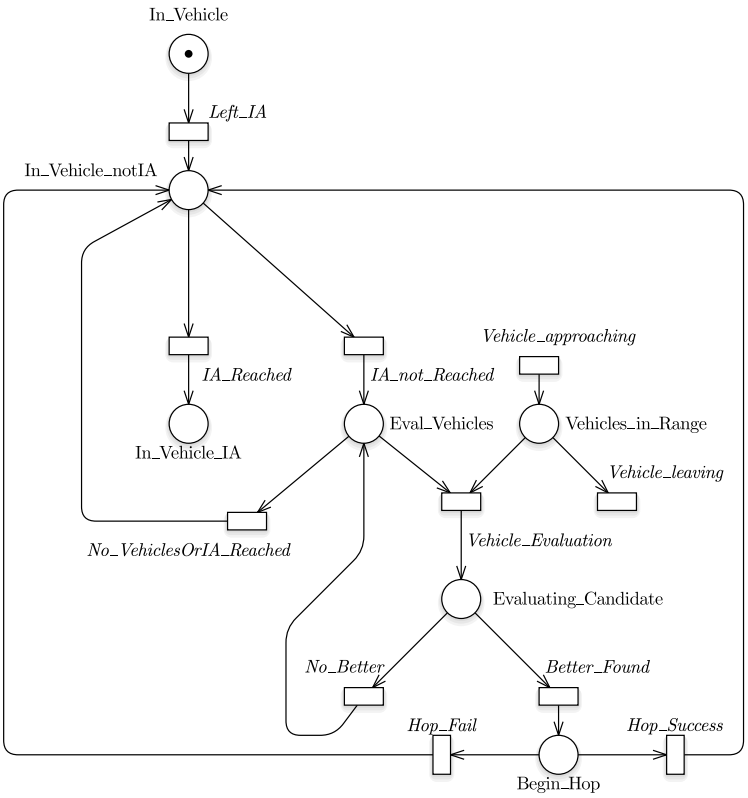


Fig. 4. Returning to the interest area after leaving it.

References

1. Ilarri, S., Delot, T., Trillo-Lado, R.: A data management perspective on vehicular networks. IEEE Communications Surveys and Tutorials, ISSN 1553-877X 17(4), 2420–2460 (Fourthquarter 2015)
2. Milojicic, D., Douglis, F., Wheeler, R.: Mobility: processes, computers, and agents. ACM (1999)
3. Olariu, S., Weigle, M.C.: Vehicular Networks: From Theory to Practice. Chapman & Hall/CRC, first edn. (2009)
4. Trillo, R., Ilarri, S., Mena, E.: Comparison and performance evaluation of mobile agent platforms. In: The Third International Conference on Autonomic and Autonomous Systems (ICAS). pp. 41–46. IEEE Computer Society (2007)
5. Urra, O., Ilarri, S., Trillo-Lado, R.: An approach driven by mobile agents for data management in vehicular networks. Information Sciences 381, 55–77 (March 2017)
6. Uzcategui, R.A., Sucre, A.J.D., Acosta-Marum, G.: Wave: A tutorial. IEEE Communications Magazine 47(5), 126–133 (May 2009)