Graphical modelling of process algebras with DrawNET

Stephen Gilmore
Laboratory for Foundations of Computer Science
University of Edinburgh

Marco Gribaudo
Dipartimento di Informatica
Università degli Studi di Torino

I. Introduction

High-level languages for performance modelling are often complex to use. Further, they are unfamiliar to the people who would most directly benefit from using them, namely practising software engineers and application developers. One approach to reducing the difficulty of using these languages is to equip them with a graphical syntax which is more intuitive than the more unfamiliar languages of process algebras and other state-based modelling formalisms.

We have extended the DrawNET modelling tool to facilitate the design of PEPA net models [1] in addition to its existing support for Petri net-based formalisms and fault trees. This work is part of an ongoing programme to extend DrawNET to support a very wide range of graphical modelling formalisms. Other candidate modelling formalisms for future incorporation include queueing networks.

DrawNET is a framework which can be used to edit models which are expressed in graph-based modelling formalisms. It is based on a two-level structure in which one level is used to describe the modelling language itself and the other is used to describe models in this language. One of the key features of DrawNET is that it is a modern software tool which utilisies current implementation technology such as Java and XML. In this way both the application code and its data are immediately portable across a wide range of systems.

The current focus of the DrawNET tool is on model composition, delegating analysis and solution of the models to other tools, such as the PEPA Workbench or GreatSPN. This is the way in which we have used DrawNET here. Using DrawNET as a model composition tool is a very appropriate use for the PEPA nets formalism in particular. Used in this way, DrawNET provides a level of abstraction from the concrete syntax which is used by the PEPA net tools and the PEPA tools. The PEPA nets modelling language is supported directly by the PEPA Workbench for PEPA nets and by the PEPA net compiler. The compiler compiles PEPA net models to equivalent models in the PEPA stochastic process algebra (which can be seen as a sublanguage of the PEPA nets language). Through the use of the PEPA nets compiler, PEPA net models can also be processed by any of the PEPA tools and, thanks to the efforts of several groups of developers and users, PEPA is supported by a wide range of tools. These include the PEPA Workbench [2] itself, but also the Möbius multi-paradigm modelling framework (see [3] and [4]), the PRISM probabilistic model checker [5] and the Imperial PEPA compiler (see [6] and [7]) which compiles PEPA models to the modelling language of DNAmaca [8]. Using DrawNET as a model editor whose output is preprocessed into the input formats of the PEPA tools allows us to eliminate the effect of small differences in concrete syntax between the PEPA tools. This is achieved by building knowledge of these small differences into the DrawNET model unparser, rather than requiring every PEPA user to master them.

The novelty in the current work is in using a graph-based tool to compose the process algebra models which are usually expressed in a textual concrete syntax. In addition, the unusual challenge afforded to DrawNET by the PEPA nets language is the hierarchical composition of sub-models which are expressed in different formal languages. A PEPA net uses a Petri net superstructure to compose related PEPA models. These PEPA models are themselves a composition of a collection of sequential PEPA components. This structure is faithfully represented in our encoding of PEPA nets in DrawNET.

II. A graphical language for process algebras

In this section we discuss the design of the graphical representation for PEPA nets in DrawNET. An important motivation for the PEPA nets language was to take advantage of the intuitive reading of graph-based formalisms such as Petri nets. In designing a corresponding graphical representation for the process algebra sublanguage we wished to keep faith both with the design of the PEPA nets modelling language and the principles on which the PEPA stochastic process algebra is founded [9]. PEPA is a compact formal language with a small number of carefully chosen concepts. To reflect this we wished to use a limited number of graphical objects with clearly defined roles.

In order to reinforce the modeller's intuition when drawing sequential process algebra sub-components we utilised the "blackboard notation" for process algebras in which $\bullet$ is used to represent a state, $\bullet \rightarrow \bullet$ represents a transition and $\bullet \leftarrow \bullet \rightarrow \bullet$ represents a choice. States are labelled by a descriptive name. Transitions are labelled by a type name describing the activity (for a queue this could be *arrive* or *serve*) and a rate characterising the exponentially-distributed delay associated with this activity. The diagrammatic syntax which we use for these sequential components can then be read literally as a drawing of the labelled transition system described by the component. In our experience of modelling with PEPA, the sequential components in typical models have a modest number of states, perhaps less than ten, so a diagram of a sequential component is practical to draw, whereas the state transition diagram of the exploded interleavings of the sequential components would have perhaps thousands or millions of states and transitions and would have been entirely impractical to draw. Other concurrent system modelling tools have adopted the same approach, for example Holzmann's SPIN model checker [10] and Kramer and Magee's LTSA [11] also represent sequential components graphically as labelled transition systems.

We then considered the problem of representing the concurrent composition of these sequential components where the composition operator both instantiates and configures the components through the use of an explicit synchronisation set. The guiding principle that we used in this problem was that we wanted to have a "high-level" representation of concurrent composition. For example, we rejected the idea that the modeller would simply have to draw the abstract syntax tree of the model. The notation which we use instead allows communicating sequential components to be connected in an unrestricted graph and is a variant of Milner's *flow graphs* [12].

Milner's flow graphs present a static structural view of the system model, showing the components of the model and their connections to other components. Milner's CCS has a typed language of capabilities with input actions matched by output actions, restricting the modeller to binary synchronisation only. Capabilities are represented graphically in his flow graph as *ports*, one for every action type. A synchronisation point is depicted by an arc connecting an input port to an output port. The PEPA language provides different synchronisation primitives from CCS, offering CSP-style multi-way synchronisation. In our experience in practical modelling with PEPA, components typically seem to synchronise over a set of several activity names, with single activity synchronisation being simply a special case of the more general use. For this reason we decided to use a variant of Milner's flow graph notation for our concurrent composition notation. We decorate each connecting arc in our diagrams with a set of activity names. This has the effect of reducing clutter in the diagram and mapping several of Milner's arcs onto one of ours.

One of the design decisions of the project concerned which elements to represent only in attributes associated with graphical elements, not as graphical elements themselves. In other work on DrawNET [13] marking-dependent rates are not depicted graphically. It would be possible to do this but embedding the marking dependency in the firing rate is preferred to adding another graphical primitive. We encode PEPA's *hiding* combinator in this way, specifying hidden activities as attributes of a component instance.

III. Implementation in DrawNET

In this section we provide more details of the implementation of the DrawNET tool and of the encoding of PEPA nets in DrawNET. The underlying theory of the DrawNET tool is explained in [14], [15], [16]. DrawNET is applied to the Fluid Stochastic Petri Net formalism in [13]. Papers on the current state of the tool are [17], [18], [19].

DrawNET categorizes the modelling primitives into three categories: nodes, edges and sub-models. The nodes and edges are the graph components and sub-models are used to give hierarchical composition in models. Each of these categories is termed an *element* of the DrawNET language. Each element can have some properties and in addition the edges can have two kinds of constraints. The first is that they may constraint the types of elements which can be connected. The second is that they can specify the cardinality of incoming edges and outgoing edges. DrawNET checks the validity of these constraints as a model is being composed and will not allow a user to break any constraint which is specified in the formalism. This incremental approach to giving diagnostic feedback on errors in models is better suited to non-expert users than is the traditional batch-mode approach. In the latter the user is allowed to compose erroneous models. These are then faulted with a series of diagnostic error message at the time of compilation into another representation.

Sub-models are containers with visibility attributes and specify constraints on which elements they can contain. Each sub-model has an associated language which may be different from the language in which the enclosing model is expressed. We can exploit this feature

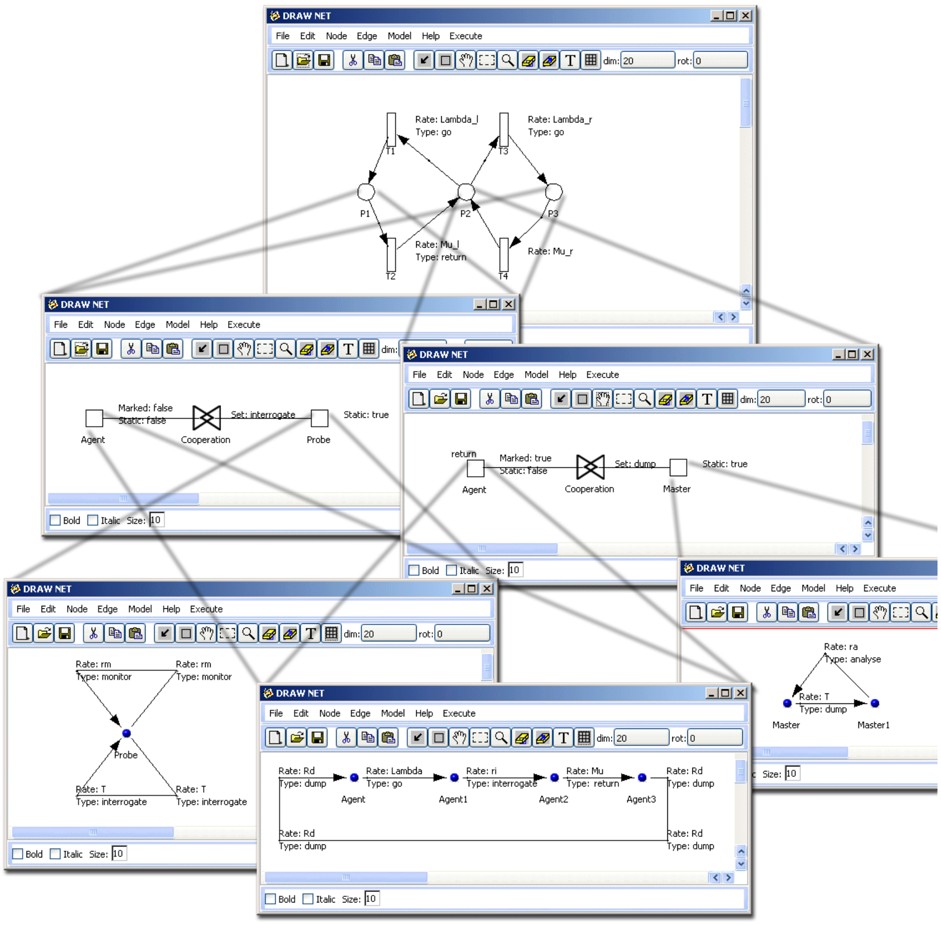

Fig. 1. A hierarchy of models in DrawNET

to allow models to be composed in hybrid modelling languages. Examples of these include combinations of Petri nets with other modelling languages such as queueing networks or process algebras. In the next section we will see an example of this multi-language composition used twice. In the first case, sequential PEPA components are composed into a parallel composition. In the second case, PEPA contexts are composed into a PEPA net.

IV. Example and Screenshots

We describe a simple PEPA net model of a mobile agent system. A mobile agent visits three places, $P1$, $P2$ and $P3$. In places $P1$ and $P3$ the agent interrogates a network probe for statistical data which it has collected on patterns of network usage. Having harvested this data, the agent returns home (to place $P2$) and dumps the gathered information to a master probe. The master probe then performs a sophisticated analysis of the data. The structure of the system allows this analysis to be overlapped with the agent's transmission time. The system is illustrated in Figure 1.

Figure 1 composes six screenshots of the DrawNET graphical interface editing the layers of the mobile agent PEPA net model. The top of the diagram shows the net structure itself in classical Petri net notation with circles for places and vertical bars for transitions. In our example this specifies three places, $P1$, $P2$ and $P3$ with the net transitions which move tokens labelled by **go** and **return**.

The centre of the diagram shows the composition layer in our diagrammatic syntax which resembles Milner's flow graphs. Here sequential component instances are composed and configured using co-operation sets. These sub-models represent contents which specify the cells and the static components at each place in the net. This notation uses the distinctive PEPA combinator (the butterfly symbol).

The composition layer of the model also differentiates static components from tokens. Only those PEPA components which are marked as being tokens can circulate around the net. Components which are static are fixed in one place and cannot move. The composition layer also encodes the initial marking of the PEPA net. Two Boolean variables are associated with each component for the purpose of recording these two facts (the Boolean variables are "Marked" and "Static"). The dependency between these is that only non-Static components can be Marked. These two variables record the three possibilities.

Finally at the bottom of the diagram we see the sequential components described by labelled transition systems using the blackboard notation for process algebra terms. Sequential components define the behaviour of the *Agent* token and the *Master* and *Probe* static components. Activities specify their rate and type.

In all, the three different graphical designs separate out the layers of the PEPA net model.

V. Conclusions and further work

We have extended the DrawNET tool to allow modellers to compose PEPA net models graphically without needing to know the details of the concrete syntax of tools such as the PEPA Workbench, Möbius, PRISM and ipc. We hope that this extension makes the language more accessible to novice users or (non-technical) readers of PEPA net models.

We found that the design of DrawNET strongly supports the incorporation of a new graphical formalism such as our diagrammatic syntax for PEPA nets. The DrawNET tool has been carefully engineered to provide generic capabilities which we have been able to reuse here.

In this paper we have described model composition but not solution. An obvious next step for further work is to integrate the solution component with the graphical user interface in order to present the results of the solution back to the user in the terms of their model.

Acknowledgments: Stephen Gilmore is supported by the DEGAS (Design Environments for Global ApplicationS) project IST-2001-32072 funded by the FET Proactive Initiative on Global Computing. Marco Gribaudo is supported by the EEC IST project 25434 DepAuDE.

References

[1] S. Gilmore, J. Hillston, L. Kloul, and M. Ribaudo. PEPA nets: A structured performance modelling formalism. To appear in Performance Evaluation, 2003.

[2] S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In *Proceedings of the Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, number 794 in Lecture Notes in Computer Science, pages 353–368, Vienna, May 1994. Springer-Verlag.

[3] G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. Webster. The Möbius modeling tool. In *Proc. 9th Int. Workshop on Petri Nets and Performance Models (PNPM '01)*, pages 241–250, Aachen, Germany, September 2001. IEEE Comp. Soc. Press.

[4] G. Clark and W.H. Sanders. Implementing a stochastic process algebra within the Möbius modeling framework. In L. de Alfaro and S. Gilmore, editors, *Proceedings of the first joint PAPM-PROBMIV Workshop*, volume 2165 of *Lecture Notes in Computer Science*, pages 200–215, Aachen, Germany, September 2001. Springer-Verlag.

[5] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. In J.-P. Katoen and P. Stevens, editors, *Proc. 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, volume 2280 of *LNCS*, pages 52–66. Springer, 2002.

[6] J.T. Bradley, N.J. Dingle, S.T. Gilmore, and W.J. Knottenbelt. Derivation of passage-time densities in PEPA models using the imperial PEPA compiler (ipc). Submitted for publication, March 2003.

[7] J.T. Bradley, N.J. Dingle, S.T. Gilmore, and W.J. Knottenbelt. Extracting passage times from PEPA models with the HYDRA tool: A case study. In S. Jarvis, editor, *Proceedings of the Nineteenth annual UK Performance Engineering Workshop*, University of Warwick, July 2003.

[8] W.J. Knottenbelt. Generalised Markovian analysis of timed transition systems. Master's thesis, University of Cape Town, 1996.

[9] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[10] G. J. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003. To appear.

[11] J. Magee and J. Kramer. *Concurrency: State Models and Java Programs*. Wiley, 1999.

[12] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[13] M. Gribaudo. FSPNEdit: a Fluid Stochastic Petri Net Modeling and Analysis Tool. In *Proceedings of Tools of Aachen 2001 International Conference on Measuring, Modeling and Evaluation of Computer and Communication Systems*, pages 24–28, September 2001.

[14] M. Gribaudo and A. Valente. Framework for graph-based formalisms. In *Proceedings of the first International Conference on Software Engineering Applied to Networking and Parallel Distributed Computing 2000, SNPD'00*, pages 233–236, May 2000.

[15] M. Gribaudo and A. Valente. Two levels interchange format in XML for Petri Nets and other graph-based formalisms. In *Proceedings of the 21st International Conference on Application and Theory of Petri Nets*, pages 22–29, June 2000.

[16] M. Gribaudo and D. Sessi. A Multiparadigm Simulation Framework. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA'01*, pages 1647–1653, June 2001.

[17] G. Franceschinis, M. Gribaudo, M. Iacono, N. Mazzocca, and V. Vittorini. Towards an Object based Multi-Formalism, Multi-Solution Modeling Approach. In *Proceedings of Second Workshop on Modelling of Objects, Components, and Agents, (MOCA2002) Aarhus, DK*, Aug 2002.

[18] V. Vittorini, G. Franceschinis, M. Gribaudo, M. Iacono, and C. Bertoncello. DrawNet++: a Flexible Framework for Building Dependability Models . In *Proceedings of Tools presentations, Proc. of the International Conference on Dependable Systems and Networks (DSN2002)*, June 2002.

[19] V. Vittorini, G. Franceschinis, M. Gribaudo, M. Iacono, and N. Mazzocca. DrawNet++: Model Objects to Support Performance Analysis and Simulation of Complex Systems. In *Proceedings of 12th International Conference on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation*, April 2002.