Modelling Large-Scale Discrete Event Systems as Modular Petri Nets using GPenSIM

Reggie Davidrajuh
Electrical and Computer Engineering
University of Stavanger
Stavanger, Norway
E-mail: reggie.davidrajuh@uis.no

***Abstract* - This paper introduces modularization of large Petri net models using the tool General Purpose Petri net Simulator (GPenSIM). The newer version of GPenSIM allows modularization so that the flexibility of the models as well as the comprehensibility can be improved. In addition, modularization reduces the development time and it increases the robustness. In the first part of this paper, following a brief literature study, the design criteria for developing modules for large Petri net models are discussed. In the second part, an application example is given. The theory and the application example shows that modularization in GPenSIM is a convenient way of modeling large systems; modularization using GPenSIM is not an extension of Petri nets, as it is not providing a new Petri net class.**

Keywords - modular Petri nets; Petri nets; GPenSIM

I. INTRODUCTION

This paper discusses modularization as a technique for improving the usefulness of large Petri net models. Though modularization is supported in many Petri net tools (e.g. CPN), it was not possible before in the tool GPenSIM. The newer version of GPenSIM allows modularization so that the flexibility of the models as well as the comprehensibility can be improved. In addition, modularization reduces the development time of large Petri net models; modularization also increases the robustness of the models.

In this paper: Section-II presents a brief literature study on modular Petri nets. Section-III explains what is meant by modularization; the main design criteria (such as composability, interface, the number of modules, etc.) for developing modules for large Petri net models are discussed in this section. Sections-IV and V introduce the modeling and simulation tool GPenSIM and its support for the modular model building. Section-VI presents an application example.

II. RELATED WORKS

The classical Petri net (*aka* ordinary Petri net, P/T Petri net) does not provide any support for modularization [1]. However, many of the Petri net extension (High-level Petri nets) support modularization. Colored Petri nets (CPN) provides features such as *substitution transition* and *fusion places* [2]. The substitution transitions are input and output ports of the modules; a substitution transition represents a module that is made up of many transitions and places. Thus, substitution transition abstracts away the details of the modules from the overall model. The fusion places are for modeling convenience; fusion places aliases for a place just to avoid too many arcs crisscrossing the model [3].

Literature also cites shared transitions as a mechanism for modularizing Petri net models [4; 5]. In this approach, the modules are developed separately, with the assumption that they are independent of each other. If the modules are to be synchronized, then shared transitions in these modules become the one and same transition, in order to perform synchronization.

A very closely related topic to modular Petri nets is the object-oriented Petri nets. In object-oriented Petri nets, each object is to be modeled separately, and their behavior and the form of communication are well defined in their interfaces [6; 7]. The object-oriented approach promotes model building in a hierarchical fashion than in a flat, collaborating and coordinating agent style, where the collaborating agents (modules) are considered as peers.

Literature review reveals several other Petri net extensions to solve problems in specific domains. Constraints-based Modular Petri nets (CMPNs) is specifically for requirement engineering domain, proposes a two-level hierarchical modeling approach. CMPNs is a fusion of modular Petri nets and object-oriented Petri nets, preserving some of the features (e.g. external interfaces, shared places, and transitions) of both [2].

III. MODULARIZATION

Modularization becomes a necessity when large Petri net models are developed. Modularization increases the flexibility of the models, as the modules can be reused in different models with minimal modification. In addition, changes in the model demand only change in the relevant modules, thus the rest of the modules are spared from modification. Modularization also increases comprehensibility of models, as the modules usually represent the functional entities of the real-world system they represent [9].

A distinct benefit of modularization of large Petri net models is that the development time is shortened due to the fact that different development groups could work independently on different modules; as the modules can be tested individually using drivers and stubs, the groups can develop the modules with less interaction with the other groups. Modularization also functions as data abstraction, as the modules hide information within [10].

A. *Criteria for Modularization*

The first criteria for successful of modularization depend on how effectively the Petri net model is divided into a number of modules that are representative of the functional units of the real-world system. This issue is known as the "model composability" and it is the capability to select and assemble components in various combinations in order to satisfy specific user requirements meaningfully [11].

The second criteria for successful of modularization are the sizable number of modules [12]. If there are too many modules, then the modules will be simpler and trivial, and connecting the modules together becomes difficult and error-prone. On the other hand, if there are only a few modules, then the modules can be too big to handle and can be complex to understand and code. Thus, the size of the modules must be appropriate in terms of complexity and the amount of expected coding.

Finally, the third criteria for successful of modularization are the existence of a well-defined interface for connecting the modules together [13]. Independent module building is entirely dependent on the availability of well-defined interface.

IV. GPenSIM

General purpose Petri net simulator (GPenSIM) defines a Petri net language for modeling, simulation, performance analysis, and control of discrete event dynamic systems. GPenSIM run on MATLAB platform [14]. GPenSIM is being used by several universities around the world; recently, a team of Australian researchers commended GPenSIM as the ideal tool for modeling and simulation of Event Graphs [15]; Event graph is the main class of Petri nets for modeling manufacturing systems. In addition to its usefulness as a simulator, GPenSIM can also be used a real-time controller [16]. Another main attribute of GPenSIM is that it implements Activity-oriented Petri nets (AOPN); AOPN is very useful for obtaining compact Petri net models when there are a large number of resources involved. GPenSIM supports many Petri nets extensions, such as the logical extensions (e.g. inhibitor arcs, transition priorities, and enabling functions) and the color extension.

In the earlier versions, modularization was not supported by GPenSIM. In the newest version (version 10), some basic support for modularization is offered. GPenSIM is developed by the author of this paper.

Implementing a Petri net model (as one holistic model, without modules) with GPenSIM usually requires four M-files:

1. Petri net definition file (DEF for short): a DEF defines the static Petri net graph. This means, a DEF simply declares the set of places, the set of transitions, and the set of connections (arcs) between the places and the transitions.
2. Main simulation file (MSF for short): MSF declares the initial dynamics (e.g. initial tokens, firing times of the transitions, available system resources, etc.) and runs the simulations. The code for plotting and printing the simulations results also are coded in this file.
3. COMMON_PRE file: This file defines the pre-conditions (*aka* 'guard functions') that an enabled transition must satisfy before firing. COMMON_PRE has the universal visibility, as all the enabled transitions are visible here.
4. COMMON_POST file: This file declares all the activities that are to be done after a transition completes firing. For example, this file may instruct a transition to release the resources it was holding during the firing. COMMON_PRE has the universal visibility, as all the transitions that complete firing are visible here.

V. Modular GPenSIM

Modules in GPenSIM is a natural extension of the unsegmented modeling approach that was the only option until recently.

A. *Implementing modules: the files*

Since there are a number of modules now instead of one holistic model, we need more DEF files, one for defining each module:

- Modules are defined as separate Petri nets using their own Petri net definition file (DEF). In addition to each DEF defining the corresponding module, one more DEF is needed for defining the inter-modular connections.

Since the transitions residing inside a module possess limited scope (module scope and not universal scope), the pre-conditions for the firing of these transitions are defined in a specific MOD_PRE file.

- MOD_PRE file: This file defines the pre-conditions for a transition residing inside a module. The transitions residing inside a module are not visible in the COMMON_PRE file. Each module can have their own MOD_PRE file.
- MOD_POST file: This file declares all the activities that are to be done after a transition residing inside a module completes firing. The transitions residing inside a module are not visible in the COMMON_POST file. Each module can have their own MOD_POST file.

B. *The Input and Output Ports*

There are no shared transitions in GPenSIM. This means it is not possible to synchronize activities (transitions) that reside in different modules. Synchronization of different modules must happen at the gates (input ports) of the modules. Hence, only the transitions that are declared as the input ports and the output ports of modules have universal visibility, thus can be seen in the COMMON_PRE and COMMON_POST files. The input and output ports of a module have module specific visibility as well since they are part of that module too.

VI. Application Example

The application example taken from [17] models flow capacity of the Harstad/Narvik Airport (EVE) in the North Norway. The Petri net model shown in the Fig.4 is a holistic model. The aim this section is to modularize the holistic model and then implement the model with GPenSIM.

The model shown in the Fig.4 clearly indicates that the airport possesses a single runway (RWY) and that it has 20 gates. Using the approach by Activity-oriented Petri nets (AOPN), the resources such as control tower, runway and gates can be eliminated from the Petri net, thus the model becomes less complicated and easy to handle. For further information on AOPN, the interested reader is referred to [18; 19].

Though there is only one runway, the Petri net shows three virtual runways. This is because, there are three types of aircraft (A/C) handled by the airport, and each type has significant differences in speed, thus take different runway lengths and time when landing and takeoff.

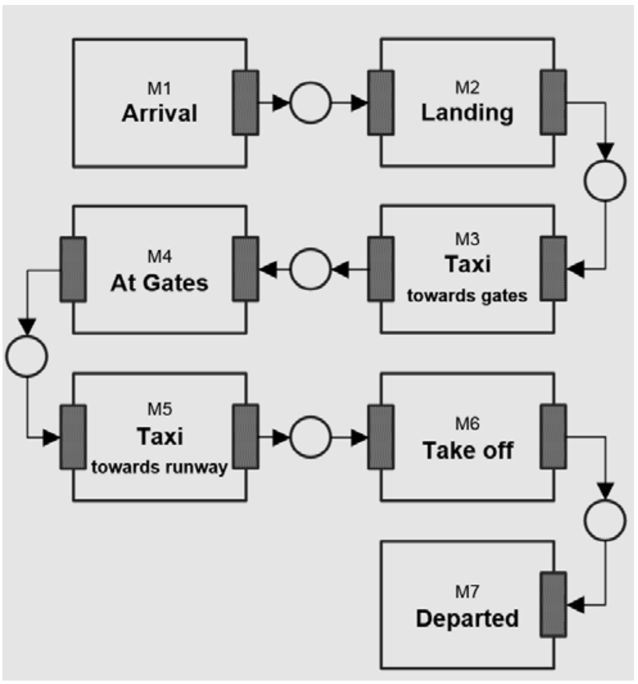


Figure 1. The modular Petri Ne model of airport flow capacity.

A. The Modular Petri Net Model

The modular Petri net model is shown in the Fig.1. In this model, all the modules represent one or more processes of the airport model for capacity flow. Further:

- The modules M1 (arrival) and M7 (departed) are the modules that have only one port. M1 is the generator of aircraft arrival, thus does not need any input port, as it is a generator. Module M7 is a sink that absorbs all the tokens representing the departed aircraft, thus does not need any output port.
- The input and output ports of the modules are transitions; buffering places are used for inter-modular connections.
- All the modules are connected serially, causing specialized interfaces between any two adjacent modules. There is no need for a generalized interface for connecting any module with the rest of the model.
- Since the modules are connected serially, duplicating a module becomes unavoidable. For example, modules M3 (taxiing from runway towards gates) and M6 (taxiing from gates towards runway) represent the same functionality; however, two instances of the module are needed because the modules play their role at different points along the serial connection.

B. The Modular Petri Net Model: an Improved Version

The improved model is shown in the Fig.2.

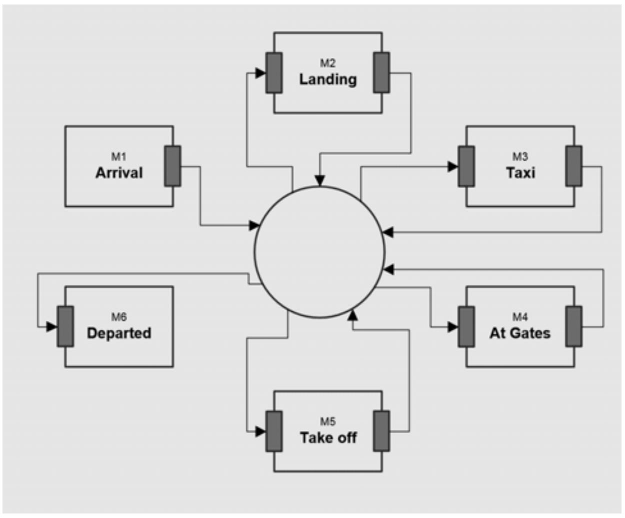


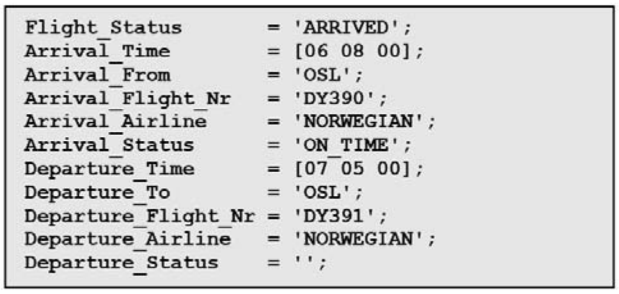
Figure 2. The improved modular Petri Ne model of airport flow capacity.

In this model:

- All the modules are connected to one another using one and the same common interface. This demands a detailed interface that is general enough to allow communication between any two modules.
- Through their input ports, the modules pull the tokens that are earmarked for them from the common buffer. After working on the tokens, the modules push the tokens back into the common buffer through their output ports. Thus, the common buffer virtually functions as the system-wide common bus for information exchange, eliminating the need for global variables.

- Due to the star-type topology of the connection of the modules, the necessity to duplicate the Taxi module (module M3/M5) becomes obsolete. The module M3 (taxi) in the Fig.2 can function for taxiing in both directions.

A sample interface for connecting the modules together for airport model is shown below in the Fig.3. In this interface, it is the first field (‘Flight_Status’) that is used by the modules for pulling the correct tokens into the module. For example, the token with value ‘ARRIVED’ means, an aircraft has just sought permission to land and thus the module for landing (M2) must process this aircraft.



```
Flight_Status       = 'ARRIVED';
Arrival_Time        = [06 08 00];
Arrival_From        = 'OSL';
Arrival_Flight_Nr   = 'DY390';
Arrival_Airline     = 'NORWEGIAN';
Arrival_Status      = 'ON_TIME';
Departure_Time      = [07 05 00];
Departure_To        = 'OSL';
Departure_Flight_Nr = 'DY391';
Departure_Airline   = 'NORWEGIAN';
Departure_Status    = '';
```

Figure 3. The interface for enabling inter-modular communication.

The Table-I given below shows the values of the field ‘Flight_Status’ and the relevant modules for processing.

TABLE I. VALUES FOR THE FIELD ‘FLIGHT_STATUS’ AND THE CORRESPONDING MODULES.

Value	*Source module*	*Destination module*
ARRIVED	M1: Arrival	M2: Landing
LANDED	M2: Landing	M3: Taxiing
TO_GATE	M3: Taxiing	M4: At_Gates
BOARDED	M4: At_Gates	M3: Taxiing
TO_RWY	M3: Taxiing	M5: Take_Off
TOOK_OFF	M5: Take_Off	M6: Departed

C. Simulations

The M-files for simulation are not given here due to brevity; interested readers are encouraged to visit the webpage [20] for downloading the codes and experimenting with them.

VII. CONCLUSION

This paper presented the facilities in GPenSIM for developing modular Petri net models. Through an application example, this paper also suggests connecting the modules together in a star topology rather than in a serial manner. Connecting all the modules in a star topology demands an interface that is general enough and well detailed to allow communication between any two modules.

REFERENCES

[1] L. Popova-Zeugmann, Time and Petri Nets, Springer-Verlag Berlin Heidelberg, 2013

[2] Yu-Jin Song, Jong-Kun Lee, “Analysis of Petri net models using transitive matrix”, Systems Man and Cybernetics 2000 IEEE International Conference on, vol. 4, pp. 3122-3127 vol.4, 2000, ISSN 1062-922X.

[3] J. Billington, S. Christensen, K. Van Hee, E. Kindler, O. Kummer, L. Petrucci, and M. Weber, “The Petri net markup language: concepts, technology, and tools”. In International Conference on Application and Theory of Petri Nets (pp. 483-505). Springer Berlin Heidelberg, June, 2003.

[4] G. Bucci and E. Vicario, “Compositional Validation of Time-Critical Systems Using Communicating Time Petri Nets”, IEEE Trans. Software Eng., vol. 21, no. 12, pp. 969-992, Dec. 1995.

[5] S. Christensen, N. Hansen, “Coloured Petri Nets Extended with Channels for Synchronous Communication”, Application and Theory of Petri Nets '94, 1994.

[6] C. A. Lakos, and C. D. Keen, LOOPN++: A new language for object-oriented Petri nets. Department of Computer Science, University of Tasmania, 1994.

[7] L. C. Wang, “An integrated object-oriented Petri net paradigm for manufacturing control systems”. International Journal of Computer Integrated Manufacturing, 9(1), 73-87, 1996.

[8] R. Esser, “An object oriented Petri net approach to embedded system design”, Doctoral dissertation, University of Adelaide, 1996.

[9] P. K. Davis, and R. H. Anderson. “Improving the composability of DoD models and simulations.” The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology 1.1, 2005: 5-17.

[10] K. Leino, M. Rustan, and G. Nelson. “Data abstraction and information hiding.” ACM Transactions on Programming Languages and Systems (TOPLAS) 24.5 (2002): 491-553. 2002.

[11] L. Cardelli, and P. Wegner, “On understanding types, data abstraction, and polymorphism”. ACM Computing Surveys (CSUR), 17(4), 471-523, 1985.

[12] Schilling, M. A. (2000). Toward a general modular systems theory and its application to interfirm product modularity. Academy of management review, 25(2), 312-334.

[13] Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. Communications of the ACM, 15(12), 1053-1058.

[14] R. Davidrajuh, “Developing a new Petri net tool for simulation of discrete event systems.” 2008 Second Asia International Conference on Modelling & Simulation (AMS). IEEE, 2008.

[15] Cameron, A., Stumptner, M., Nandagopal, N., Mayer, W., & Mansell, T. (2015). Rule-based peer-to-peer framework for decentralised real-time service oriented architectures. Science of Computer Programming, 97, 202-234.

[16] R. Davidrajuh, “Developing a Petri Nets based Real-Time Control Simulator”. International Journal of Simulation, Systems, Science & Technology (IJSSST), 12(3), 28-36, 2012.

[17] R. Davidrajuh, and B. Lin, “Exploring airport traffic capability using Petri net based model”, Expert Systems with Applications, 38(9), 10923-10931, 2011.

[18] R. Davidrajuh, “Activity-Oriented Petri Net for scheduling of resources”, In 2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC) (pp. 1201-1206). IEEE, October 2012.

[19] R. Davidrajuh, “Modeling Resource Management Problems with Activity-Oriented Petri Nets”. In Computer Modeling and Simulation (EMS), 2012 Sixth UKSim/AMSS European Symposium on (pp. 179-184). IEEE, November 2012.

[20] Modular Petri net model for airport flow capacity, Available from: http://www.davidrajuh.net/gpensim/2016-AMC-3-Modular-PN

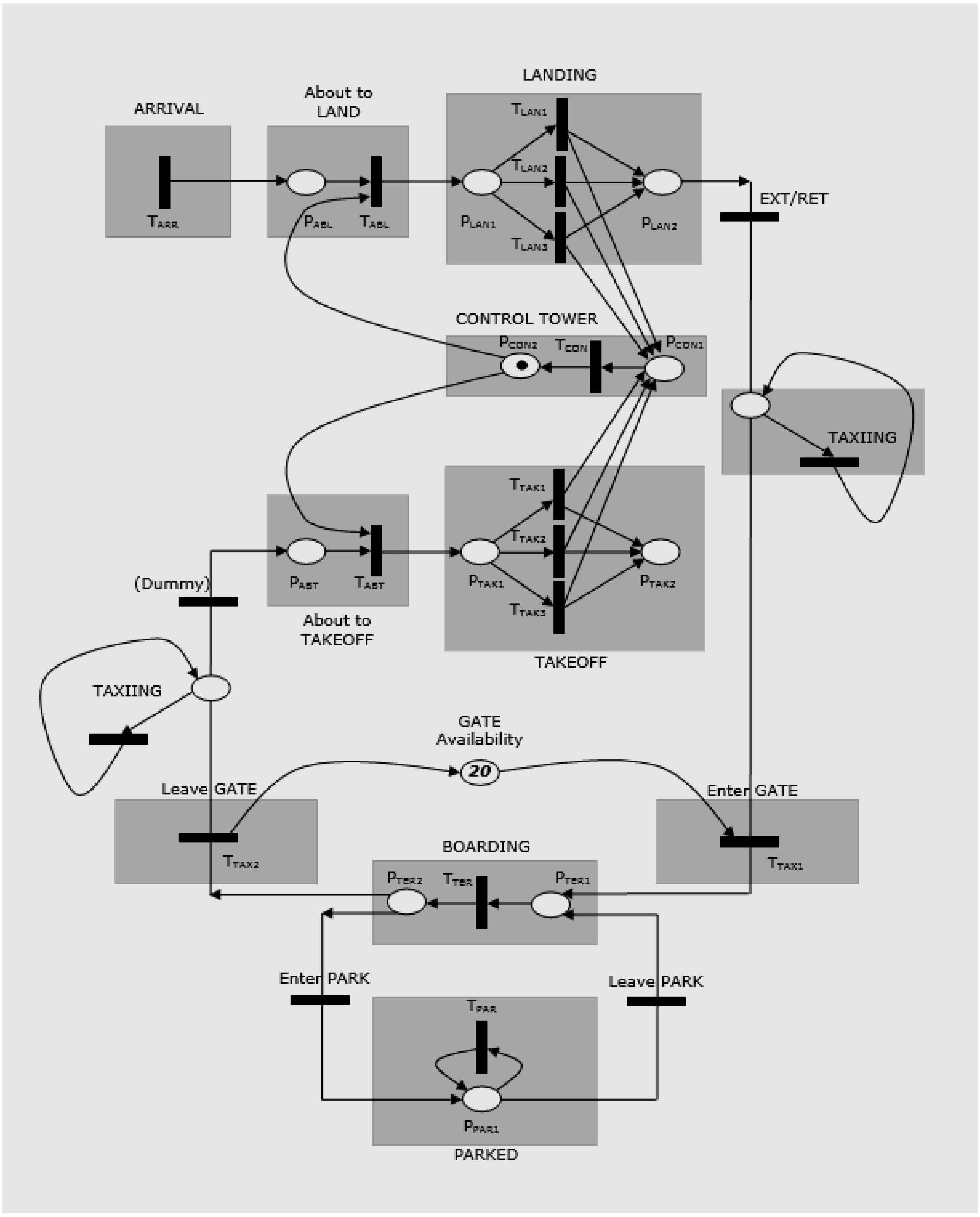


Figure 4. The holistic Petri net model for airport flow capacity (adapted from [17]).