A Conceptual Modeling Technique for Discrete Event Simulation of Operational Processes

Henk Jan Pels and Jan Goossenaerts
Technische Universiteit Eindhoven, Faculty of Technology Management,
Postbox 513, 5600 MD, Eindhoven, The Netherlands, h.j.pels@tue.nl

Abstract

A formal modeling technique, based on colored timed Petri net and UML static structure modeling languages is used to teach students to model their business process problem as a discrete event system, before they build a working simulation model in a simulation tool (in our case Arena). Combining Petri net and UML static structure diagrams, one can build an abstract, well defined and complete model. This model enables the simulation analyst to make an unambiguous, complete and yet easily readable model of the target operational process. The two most important classes of decisions that are reflected in the conceptual model are the choice of the real world details to be taken in or left out the model and the precise specification of the output parameters of the simulation. This paper describes the modeling technique and discusses its value in teaching and in the formulation of decision problems regarding operational processes.

Keywords

Discrete Event Simulation, Conceptual Modeling, Computer Independent Model, UML, Petri Nets.

1 Introduction

Model-driven systems development is gaining importance. The models at the three OMG MDA layers (Miller and Mukerji, 2003) (computation independent, platform independent and platform specific) matter in different development phases, each of which offers its own contribution to the reduction of risks and to the system design (Dick & Chard, 2003). The Computation Independent Model (CIM) shows the system in the environment in which it will operate, and thus helps in presenting exactly what the system is expected to do. Useful as an aid to understanding a

problem and for communication with the stakeholders, it is essential to mitigate the risks of addressing the wrong problem, or disregarding needs. These risks matter also in simulation studies (Law and Kelton, 2000, Chapter 5). By articulating the CIM a simulation project can be precisely scoped and the project results can be prepared for acceptance. The Platform Independent Model (PIM) describes the system in reference to a particular architectural style (e.g., agent based or client/server) but does not show details of platform use. Its structure may be quite different from the structure of a CIM of the same system. The Platform Specific Model (PSM) is produced from the PIM or the CIM by transformation. It specifies how the system makes use of the chosen platform and technologies. In our setting the executable model for the Arena tool is compared to a PSM model. It is obtained by a rather systematic mapping from the CIM model, taking into consideration the specifics of the simulation tool.

2 Research problem

The research problem of this paper is how to make a formal and precise discrete event process model, preferably on basis of proven modeling techniques. Operational processes are built out of a number of interacting parallel processes. Process algebras exist for formally specifying such systems, but such languages are not part of the industrial engineering curriculum. Also, these languages are not suitable for communicating with the stakeholders about the operational processes.

In our course on Simulation of Operational Processes, we require that students first make a Computer Independent Model, to define the problem and the proper level of detail, have this model approved by the teacher and after that implement it in Arena. The problem so far was that the available languages, ExSpect and CPN-Tools, resulted in models either too complex and with too many student specific design choices to be verified with reasonable effort, or models at a too generic level to expose the design choices that matter.

3 Approach

Petri nets (Jensen, 1992) are well defined and have been proven to be suitable for modeling the layout of processes as well as the synchronization between parallel processes. Their visual nature facilitates communication with stakeholders. By adding time and color they support the modeling of discrete time behavior. Several executable language systems are available to specify and run timed colored Petri nets [ExSpect (van Hee, 1994), CPN (Jensen, 1992)]. Our experience with ExSpect is that it heavily relies on a functional programming language for specifying fire conditions and state changes. Our Industrial Engineering students are not able to express themselves in such a language, and often stakeholders will not understand the resulting expressions. However, without the specification of fire conditions and state

changes the Petri net model is incomplete and ambiguous, and therefore it cannot be checked for correctness (almost any graph can be good).

CPN tools provides a complete graphical language, but it appears that this language requires to specify additional places to implement state variables that are not really tokens. Also the functional programming style of specifying pre- and post-conditions and the tool-specific approach to the specification of the state variables is demanding for students with little programming skills. As a result the graphical models often loose visual resemblance with the operational process and their annotations become quite complex as an integral part of the model. Though the models are executable, their validity for a specific problem is difficult to confirm.

These reasons, and the fact that our students do understand UML static structures for the modeling of the domains in which operational processes affect system states have prompted us to use UML class and object diagrams to specify all necessary state variables in terms of such a diagram. Hence, a UML static structure diagram is used to model the state space of the operational process, while the transitions in the Petri Net model are used to specify state changes.

Our idea was to annotate each transition with a specification of its state changes in terms of logic expressions over the UML static structure. CPN-tools and UML are chosen because they are already part of the curriculum. Moreover, UML is a de facto standard in information modeling.

4 Design

In this section we explain how we build a complete discrete event model that is computation independent and unambiguously specifies a system and its behaviour. The following viewpoint specifications are used:

1. the process flow is specified as a Petri net constructed with CPN-tools,
2. the state space is specified using a UML static structure diagram,
3. the initial state of the system is specified as a marking of the Petri net with instances of the classes in the UML static structure diagram,
4. for each transition a pre and post condition is specified using predicate logic and an extension of UML OCL,
5. output variables and eventual additional functions are specified in terms of UML OCL.

4.1 Process Flow

The process structure is modeled using a timed, colored Petri net. Figure 1 shows the model for a simple queuing system, using the notation of CPN [Jensen, 1992]. The transition Arrive models the arrival of a client, who joins the queue modeled as place Wait. At the same time the next client is prepared in place NewClient. When a free server resource is available in place Free, the serving operation will start by putting the client-server pair in place Serve. The end of the operation is modeled as transition EndServe, which puts the client in place Served and returns the server in place Free.

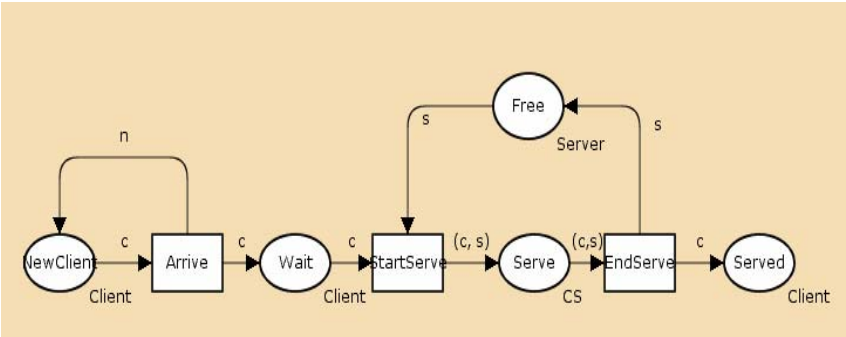

Fig. 1. Simple queuing system with client generator.

4.2 State Space

In a colored Petri net the tokens are objects with attribute values. Modeling UML static structure diagram is a very well suited tool to model the different types of orders and resources as object classes with specific attributes. Associations between object classes enable to relate attribute values of different objects in formal expressions. UML object constraint language (OCL) provides the necessary constructs to specify pre- and post-conditions in terms of attribute values (Warmer & Kleppe, 1999; OMG 2005). An advantage of this approach is that any expression can refer to any attribute value in the total state of the model. This makes an important difference with purely Petri net based languages like CPN-Tools that only supports references to attributes of tokens in the input places of the particular transition. This forces the modeler to model auxiliary places and maintain additional tokens and attribute values to make certain state variables accessible from a particular places.

Figure 2 shows the UML static structure diagram that specifies the state space for the Petri net in figure 1. A Petri net consists of Tokens and Places. The association between Token and Place expresses that every Token is always in one Place. Typical for Discrete Event Models are the object classes Clock and Random. A clock has an attribute time, which is supposed to increment implicitly. The class Random holds the concepts of random generator and random functions as used in stochastic simulations. These four classes are generic for every simulation model.

In the example, tokens can be either Clients or Servers, each having the attributes required to model the essential characteristics of the specific problem to be modeled. On the bottom row the objects are specified that populate the initial state of the system.

According to CPN-Tools every place has a type and consequently we label each place in the Petri net with an object class name from the UML static structure diagram.

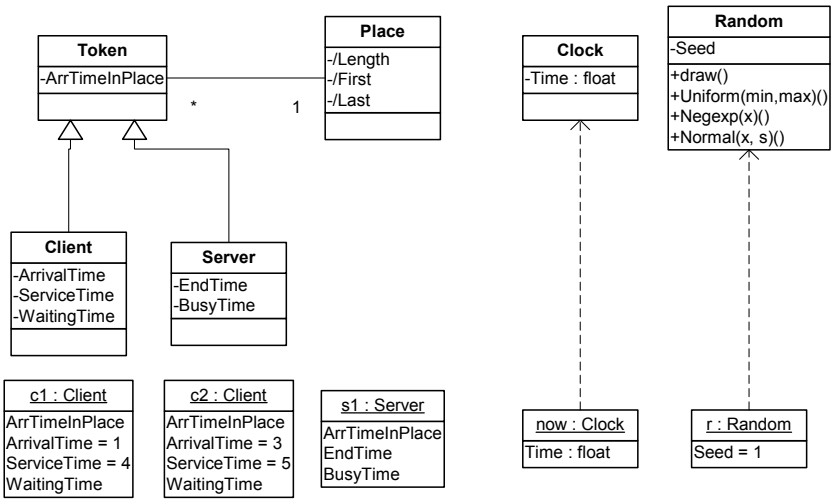

Fig. 2. A UML static structure diagram for simple waiting line simulation

The model shown above is very simple. In the problems we feed to our students complexities occur like multiple servers, multiple rows, clients that leave or switch rows under specific conditions, server failures, servers shared by different processes etc. All these situations can be elegantly modeled. However the size limits of this paper do no allow showing examples.

4.3 Transitions

From ExSpect we took the mechanism to specify transitions in terms of pre and post conditions. By default a transition fires when a token is available in each of its input places. The pre condition adds additional conditions these tokens and/or other objects must satisfy in order to allow the transition to fire: a transition fires if and only if there exists a set with exactly one token from each input place for which the pre condition holds. The clock object enables to make pre conditions time dependent.

The post condition must hold for the system state after the transition and thus enables to specify the change of system state because of the transition. The pre- and post condition are expressed in predicate logic using OCL to refer to specific attribute values,

The labels on the arcs are lent from CPN-Tools and are used as variable names in the condition specification. The labels of input arcs refer to the tokens selected for firing. The labels of output arcs refer to tokens put in the respective places and enable to specify changed attribute values. The following shows the specifications for transitions of the Petri net in figure 1 in combination with the state space specified in figure 2:

```
Init
     c1.Place = NewClient  ∧
```

```
        s1.Place = Free ∧
        s1.BusyTime = 0;

    Arrive
    Pre    c.ArrivalTime = now.Time
    Post   Wait <- c    ∧
        NewClient <- n    ∧
        n.ArrivalTime := now.Time+r.Negexp(5)   ∧
        n.ServiceTime := r.Uniform(3,6);

    StartServe
    Prec = Wait.First
    Post   Serve <- (s, c)   ∧
        s.EndTime := now.Time + c.ServiceTime      ∧
        c.WaitingTime := now.Time - c.ArrivalTime    ∧
        s.BusyTime := s.BusyTime + c.ServiceTime;

    EndServe
    Pre    now.Time = s.EndTime
    Post   Served <- c    ∧
        Free <- s;
```

The block labeled with Init specifies the initial state of the system. In particular it specifies the places of the initial tokens. Initial tokens can be specified using UML as well as in the Init block. As an alternative, the places of the initial tokens can also be specified as markings in the Petri net. More often more than one of the Petri net, the UML or the transitions allow to specify some detail. In those cases the rule is that redundancy is to be avoided.

Note that the expressions are predicate logic and not programming statements. Proper interpretation of the post conditions requires some additional semantics because they must allow referring to both the old and the new state:

- Any variable that is not mentioned in the post condition remains unchanged,
- Where the <- operator (token is in output place) is used, all variables in the right operand refer to the old state, while the left operand is the place that holds the token resulting from the right operand,
- the := operator is used to specify that the left operand has, in the new state, the value resulting from the right operand, in which all variables refer to the old state,
- in all other cases variables refer to the new state.

In all small problems modeled so far we found these constructs able to express sufficient detail. For larger models, hierarchical Petri nets and place-connectors can be used.

4.4 Output Variables

Simulation models are to be used in experiments that yield output values. In order to specify the desired outputs we use functions, expressed in OCL. The expression below refers to the model of figures 1 and 2.

```
Functions
    NrServed = Served.Length;
    AverageWait = ∑(c ∈ Served: c.Waitingtime) / NrServed;
    Occupation = s1.BusyTime / now.Time * 100;
```

The expression AverageWait for the average waiting time of clients shows how expressions can reason over sets of objects. Note that objects are persistent: once created they remain in the state space. The place Served, for instance, holds at any point in time all clients served so far. In the expression for Occupation, s1 is the object name for the server, as defined in the UML model.

Functions can also be used as macros to simplify complex expressions in pre and post conditions.

5 Mapping to an Arena Executable Model

Once the Conceptual Model (CIM) has been created, and its validity (validation during early project phases) has been confirmed with the problem owners, it is important to carry the validity and relevance of the CIM to the executable model. This is achieved by the mapping approaches explained below, and finally confirmed by the verification.

The state-space and causal or flow logic expressed in the CIM must be mapped into a model that uses the ARENA building blocks. The "Petri-net token game" must be mapped on commands such as those associated with the use of resources (Seize, Delay, Release), and the life cycle of the token (e.g., Create, Dispose).

To document the derivation of the executable model from the conceptual model, it is convenient to use two tables:

- a table to map the classes and instances of the UML information model to Arena constructs
- a table to map the Petri net process model and the transition specifications to an Arena process model.

Furthermore each transition specification must be mapped to suitable specifications of the ARENA building blocks:

- for each transition:
  - identify corresponding module(s) in Arena
  - check post conditions,
- for each multi input transition:
  - check proper resources,
- for each pre condition
  - check corresponding priority rules or hold modules.

Students, who must mutually verify each other's models on consistency between the CIM and the Arena implementation, use this mapping.

6 Evaluation and Conclusions

Use of this approach in several courses has shown that students are able to construct and validate operational process models, from which teachers can easily recognize the problem and check correctness of the model.

So far the expressive power of this modelling approach appeared to be sufficient for the problems we give our students to solve. However this does not prove that the language is able to model every possible operational process situation. One apparent problem concerns the rules for reference between old and new situation in the post conditions. They may be not able to express certain complex transitions or they may become ambiguous in such situations.

Another problem is the use of OCL. Our students get only an elementary introduction, while the OCL definition is quite extensive. When using associations in expressions the result may be a set of objects. When several associations are followed in sequence, the result is a nest of sets. OCL follows the rule to flatten those nests, so that a simple set of objects results. However one must remain careful in evaluating such expressions.

Teaching discrete event simulation can benefit from the use of model-driven systems development techniques during the early project phases. The principles of model driven architecture can be applied to derive platform specific executable models from conceptual models that are precise and convenient to support communication with the problem owner. Combining Petri Net and UML static structure diagrams, one can build abstract, well defined and complete models of operational processes.

Further work will include the extension with hierarchy.

References

J. Dick, J. Chard, Requirements-driven and Model-driven Development: Combining the Benefits of Systems Engineering, Telelogic White Paper, www.telelogic.com, 2003.

K. Jensen, Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. EATCS Monographs on Theoretical Comp. Science, Springer-Verlag, Berlin, 1992.

A. M. Law and W. D. Kelton, Simulation, modeling and Analysis. Third edition. McGraw-Hill series, 2000.

J. Miller, J. Mukerji (eds.) MDA Guide Version 1.0.1, OMG, Object Management Group, 2003.

K.M. van Hee, Information Systems Engineering: A Formal Approach, Cambridge University Press, Cambridge, 1994.

J. Warmer,A. Kleppe, A., The Object Constraint Language: precise modeling with UML, Addison-Wesley, 1999.

Object Management Group (OMG), OCL 2.0 Specification. OMG document ptc/2005-06-06, June 2005.