A Comparison of Petri Net and System Dynamics Approaches for Modelling Dynamic Feedback Systems

Jim Duggan,
Department of Information Technology,
National University of Ireland, Galway,
Galway,
Ireland.
Phone: 353-91-493336
Email: jim.duggan@nuigalway.ie

Abstract

Petri nets are a valuable tool that can be used to simulate workflow systems. They are based on a state-transition approach, and in common with discrete event simulation, events can be scheduled to fire at different time intervals. This is in contrast to the stock and flow approach of System Dynamics, where workflows are aggregated and state transitions modelled continuously through sets of integral equations. This paper continues a theme explored at the Boston 2005 conference, where a paper was presented that identified a common simulation problem, and presented solutions using both discrete event simulation and system dynamics. The benefit of this approach is that it can provide modellers from different methodological worldviews insights into how common problems may be addressed. The problem approached here is a one-actor model of the Beer Game. Two models are developed, detailed experimentation is performed, and overall results and conclusions presented.

Introduction

A simulation is "the imitation of the operation of a real-world process or system over time," and involves "the generation of an artificial history of a system, and the observation of that artificial history to draw inferences concerning the operating characteristics of the real system" (Banks et. al 1999). Simulation is used to assist decision makers cope with real-world complexity, and to inform design decisions in advance of implementations that involve significant commitments of resources. This real world complexity has often been categorised into two varieties (Senge 1990): *detail* complexity and *dynamic* complexity. Sterman (2000) comments that detail complexity is dependent on the number of components in a system, or the number of combinations one must consider in arriving at a decision. An example of this could be the problem of optimally scheduling an airline's flights and crew. Such a task has a high level of combinatorial complexity, and advanced optimisation algorithms may be designed to find the best solution out of a huge range of possible alternatives. Dynamic complexity, which is present in many natural and human systems, is determined by the causal relationships, positive and negative feedbacks, and time delays in a given system, and can give rise to unexpected and counter-intuitive behaviours. The Beer Distribution Game (Sterman 1989) is a good example of how dynamic complexity can arise as a result of agent interactions.

Historically, the discrete-event simulation (DES) and system dynamics (SD) approaches have developed as two separate and distinct fields. Morecroft et. al. (2005) acknowledge that while it is apparent that most analysts opt for the method with which they are most familiar, there "appear to be very few studies that compare SD and DES, let alone give guidance on which approach might be most appropriate in different circumstances." Work that has compared both approaches includes Brailsford and Hilton (2000), where the main differences highlighted include the observation that for DES, objects in the system are distinct entities, whereas for SD such entities are aggregated into a population and treated like a continuous quantity. In addition, for a discrete model state changes occur at discrete points in time, while state changes in SD occur continuously.

Lane (2000) adds to our understanding of these differences by highlighting that the DES approach is analytic, where the emphasis is on detail complexity, while the SD approach is holistic, with an emphasis on dynamic complexity. He also comments that – on the resolution of models – DES focuses on individual entities, attributes decisions and events, while SD concentrates on homogenised entities, continuous policy pressures and emergent behaviour. A further useful distinction offered here is that while SD focuses on strategic problems, DES – perhaps because of its primary focus on detail complexity – is targeted more at operational problems.

While the goal of each simulation approach is broadly similar – each is intended to help decision makers draw inferences from a system's "artificial history", clearly, as well as highlighting the differences and similarities between them, Morecroft et. al. (2005) pose what is arguably the most challenging question of all, namely: "which should be used in a specific circumstance?"

The goal of this paper is to address this question, not in a direct way, but by providing the reader with a widely-known and understood model that is built using both DES and SD. The DES approach is based on timed Petri nets, which retain all the necessary characteristics for discrete event simulation, including randomness of events, modelling of individual entities, an event calendar and the modelling of workflow, decisions and operational constraints. The further advantage of using Petri nets is that, like system dynamics, they have an underlying rigorous mathematical basis, and several of their key concepts map on well to stock and flow representations.

The model developed is a single-actor model of the beer game, which is a straightforward model, and has the welcome benefit of replicating key behaviours associated with systems that exhibit dynamic complexity. It is anticipated that by providing the reader with these two complimentary models, along with observations and results, they will be in a position to better answer the question, as posed at last years conference, which technique should be used in a specific circumstance.

Overview of Petri Nets

Petri nets were originally developed to model causal relationships between asynchronous components of a computer system. They are also widely used for modelling workflow systems. A Petri net is made up of five elements (Bauer et. al. 1991):

1. A set of places (P). A place represents a system state, and the set of all places represents the overall system state. From a system dynamics perspective, places are closely related to stocks, as the set of stocks in a system dynamics model also represents the overall system state. Places are represented by circles in a Petri net diagram, and places contain tokens.

2. A set of transitions (T). Transitions are the actions that take place in a system, and these actions cause a change in system state, by modifying the input places, and changing the output places. At one level, this is similar to the role played by flows in a stock and flow model (the main difference is that while flows operate continuously, transitions occur at discrete and uneven time intervals). Transitions are represented by horizontal lines in a Petri net graph.

3. An input function (I), which contains the set of input places for a transition.

4. An output function (O), which contains the set of output places for a transition.

5. A set of tokens. Tokens represent entities that flow through the Petri net. For example, in a production-distribution scenario, tokens would represent individual customer orders and also resources used to add value to those orders. Tokens reside in places, and are moved around the Petri net as transitions are enabled and fire. Formally, the assignment of tokens in a Petri net is known as its marking.

The following Petri net model (figure 1) represents a capacity constrained order processing function. There are two transitions in the model (table 1), and four places (table 2).

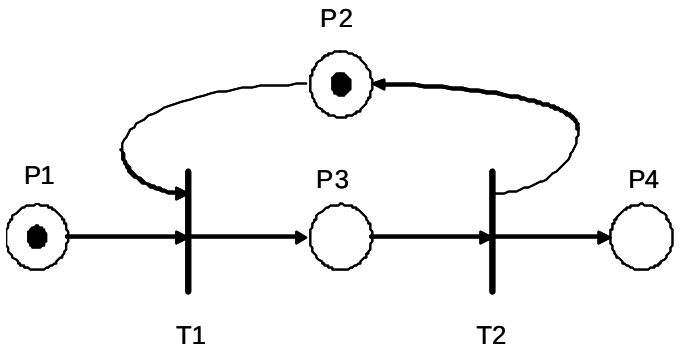

Figure 1: Petri net graph of capacity-constrained order processing function

Number	Name	Input Places	Output Places
1	Start Processing Order	P1, P2	P3
2	Finish Processing Order	P3	P2, P4

Table 1: Petri net transitions, with input and output places

Number	Name	Represents	Initial State
1	Orders Queuing	Orders	1
2	Available Operators	Operators	1
3	Orders Being Processed	Orders	0
4	Orders Completed	Orders	0

Table 2: Petri net places, with initial states

A Petri net graph models the system structure, and also indicates its state at a certain moment in time. In figure 1, there are two transitions, but in order for a transition to fire (i.e. an event is simulated), all of its input places must have at least one token. This is a useful property of Petri nets, because they can explicitly model resource constraints and so a resource-constrained transition will not occur unless the resource in present. In this

case, an order will not start unless (a) an order is in the queue (P1) and (b) an operator is available (P2) to process this order. By adding more tokens to P2, the model can be easily modified to add more capacity to the system.

A transition with a token in each of its input places will "fire", and one token will be removed from each of its input places, and a single token will be added to all of its output places. This is shown in figure 2, where P3 now has a token and each of the input places (P1, P2) have one less token. Figure two now represents the new system state, where a single order is being worked on, no operators are available, and there are no outstanding orders in the queue.

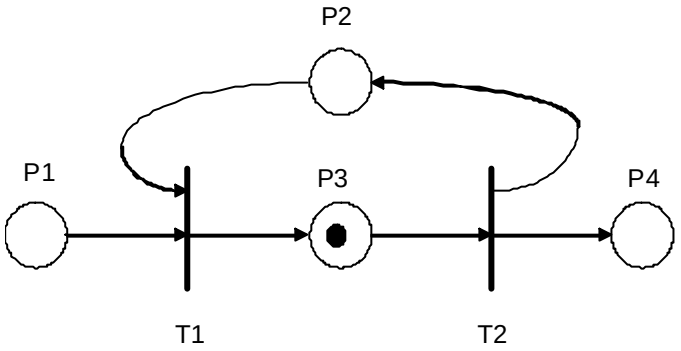

Figure 2: Petri net graph after T1 fires

With a token now in place P3, and given that T2 only has one input place, it is now clear that the next transition that is enabled to fire is T2 (Finish Processing Order). When T2 fires, it removes the token from place P3, and adds one to each of its output places, namely P2 and P3. This new state is shown in figure 3, which indicates that the operator is free again to work on the next order, and that the number of orders completed in the system is incremented by one.

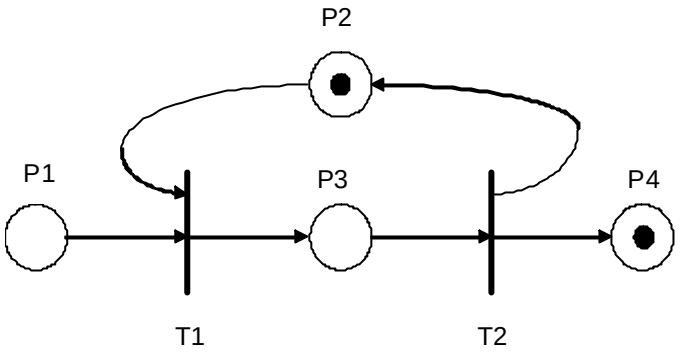

Figure 3: Petri net graph after transition two fires

While these Petri net models are useful in terms of representing the cause and effect relationships in the system, and also allowing the explicit modelling of resource constraints, a potential drawback – especially if the problem domain is workflow in service or manufacturing industries – is that there is no concept of time in these models. True, cause and effect are present, but the requirements of real-world simulation move beyond this, and in order for decision makers to gain maximum benefit from these models, some consideration must be given for the passage of time. Namely, how are event that are causally related separated in time – so, for example, in this case, how long did a token spend being processed.

Timed Petri nets were introduced to fulfil this requirement, and the mechanism employed was similar to that used in conventional discrete event simulation. For timed Petri nets, for a transition to fire, not only must it have a token in each of its input places, but it also must be *scheduled* to fire at a given *clock time*. In this example, when an order arrives in the system, two things would happen. First, a token would be placed in P1. Second, the transition T1 would be scheduled to fire at the next clock time. The internal "clock" of the simulation is always forwarded to the time of the next scheduled transition, and means that – unlike the system dynamics approach – time jumps forward in uneven measures, and only needs to move on to the next significant event (see figure 4).

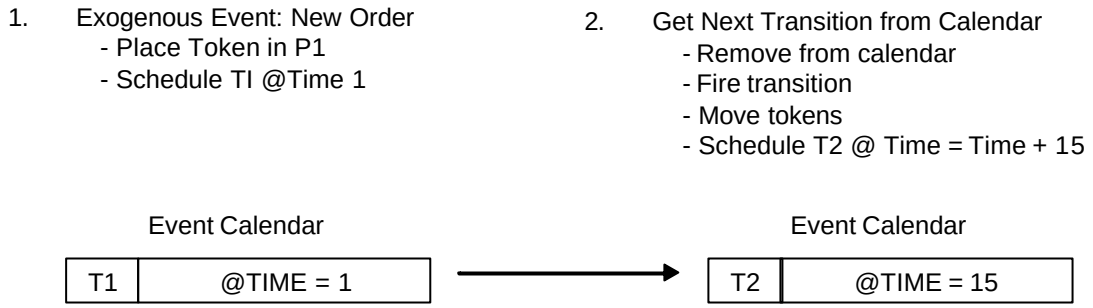

Figure 4: Using an event calendar to schedule transitions

If it happens that when a transition is scheduled but all its input places do not have tokens, then the transition will be rescheduled for the next clock time, and this rescheduling will continue until whatever resource is required has been freed up, and all the input places have at least one token. Using timed Petri nets also requires that the cause and effect between transitions must be explicitly defined, and also the average time, along with the time distribution, needs to be specified.

In summary, Petri nets allow a system to be modelled in terms of its states and transitions between states. Enhancing a Petri net with a timing mechanism means that timed Petri nets can then be deployed as a discrete-event simulation approach, which enables decision makers to model workflows, constraints, and calculate important timing statistics for each simulation run. An example of this will be described in the next section, but before that, a brief summary of the well-known system dynamics model of the single-actor beer game is presented.

The Single Actor Beer-Game Model

Typically, the beer game model (Sterman 1989) comprises four main actors: Retailer, Wholesaler, Distributor and Factory. However, the dynamics of continuous oscillation – caused by the policy where the supply line is ignored – can be replicated with a single actor (Sterman 2000). For the purposes of this paper, and in order to simplify the model while retaining the ability to replicate key dynamics, a single actor model is used, and this is modelled using the standard stock and flow model, and a Petri-net based model.

The Stock and Flow Model

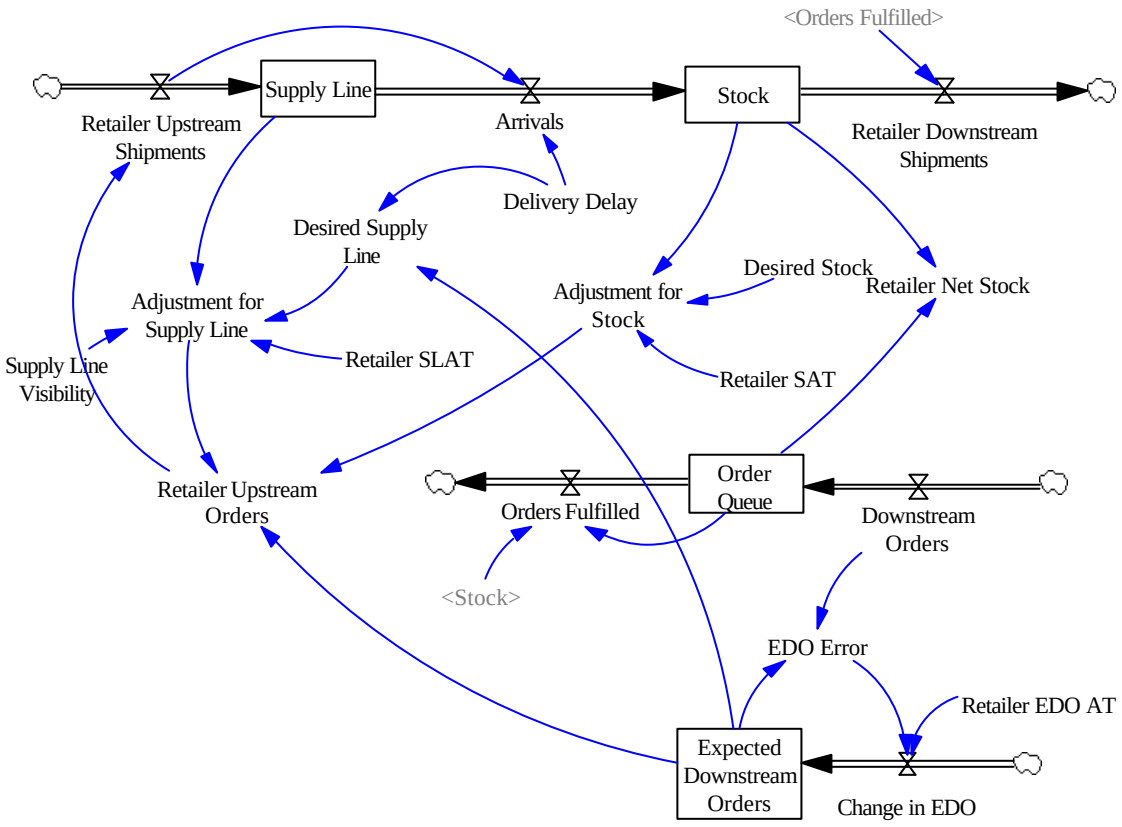

Figure 4: The single actor beer game

The main stocks in the system are the: order queue (1), retailer's stock (2) and retailers supply line (5). The decision maker's expectation for future demand is represented in (4), and the net stock is the difference between (2) and (1). The initial value of all stocks is zero, in order to align it fully with the initial state of the Petri net model.

(1) Order Queue = INTEG(Downstream Orders - Orders Fulfilled , 0)
(2) Stock = INTEG(Arrivals - Retailer Downstream Shipments , 0)
(3) Retailer Net Stock = Stock - Order Queue
(4) Expected Downstream Orders = INTEG(Change in EDO , 4)
(5) Supply Line = INTEG(Retailer Upstream Shipments - Arrivals , 0)

Customer demand (6) is modelled – along conventional lines – as a step function that doubles after twenty five time units. The order fulfilment rate (7) is constrained by the amount of stock available, and the net shipments from stock (8) is the same as this order fulfilment rate.

(6) Downstream Orders = 100 + step (100, 25)
(7) Orders Fulfilled = min (Order Queue , Stock)
(8) Retailer Downstream Shipments = Orders Fulfilled

Arrivals (9) to the system follow a fixed delivery delay (3), and the desired stock level (11) is constant for 50 time units, and is arbitrarily increased by 200 after that. The reason for increasing the goal is to provide an additional "challenge" for assessing the behaviour of the Petri net model.

(9) Arrivals = DELAY FIXED (Retailer Upstream Shipments ,Delivery Delay , 0)
(10) Delivery Delay = 3
(11) Desired Stock = 400 + step (200, 50)

The adjustment for stock (12) is the difference between the goal (11) and the current state (2), adjusted by the stock adjustment time (13). The desired supply line (14) – which represents the ideal steady state number of orders in the supply line – is the product of the deliver delay (10) and expected orders (4). The adjustment for the supply line is the supply line goal (14) minus the current supply line state (5), adjusted by the supply line adjustment time (17). A boolean variable, supply line visibility (16), is used switch this adjustment on or off, depending on whether the decision maker is aware of all feedbacks operating in the system.

(12) Adjustment for Stock = (Desired Stock - Stock) / Retailer SAT
(13) Retailer SAT = 3
(14) Desired Supply Line = Delivery Delay * Expected Downstream Orders
(15) Adjustment for Supply Line = Supply Line Visibility * ((Desired Supply Line - Supply Line) / Retailer SLAT)
(16) Supply Line Visibility = [0 | 1]
(17) Retailer SLAT = 3

Finally, the total upstream orders (21) cannot be less than zero, and are the sum of the adjustment for stock, supply line adjustment and expected downstream orders.

(18) EDO Error = Downstream Orders - Expected Downstream Orders
(19) Change in EDO = EDO Error / Retailer EDO AT
(20) Retailer EDO AT = 1
(21) Retailer Upstream Orders = max (0, Adjustment for Stock + Adjustment for Supply Line+ Expected Downstream Orders)
(22) Retailer Upstream Shipments = Retailer Upstream Orders

The Petri Net Model

The Petri net model of the single-actor beer game is shown in figure 5. A number of enhancements are made to the convention Petri net approach in that transitions T8 and T9 are special transitions that represent the reordering heuristic of the retailer, and the exogenous event that creates customer demand. Transition T8, which is scheduled to fire at the end of each day, uses the place values of P4, P5 and P6 in order to calculate the correct amount to reorder. The heuristic used is exactly the same as that specified in equation (21), and a flag can also be set to either activate or deactivate the supply line visibility. Therefore the dashed lines represent the information feedback that allows the supply line and the stock to be controlled.

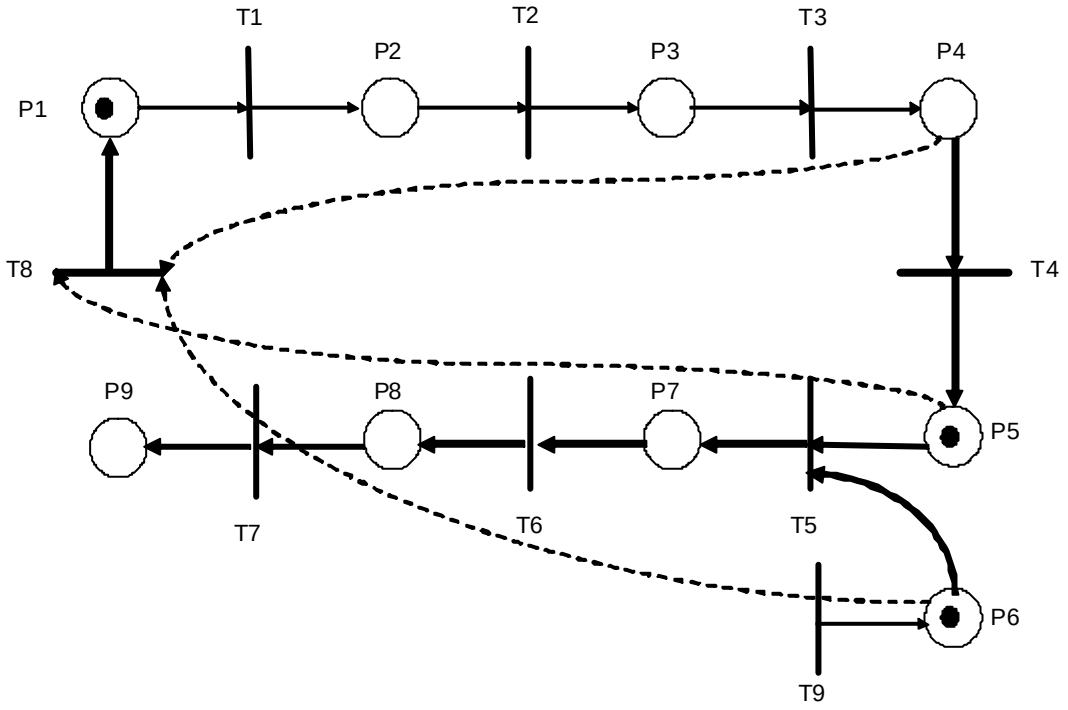

Figure 5: Petri net model of single actor beer game

The full list of transitions is shown in table 3, along with the average and standard deviation of the times (in minutes) between a "cause" transition and an "effect" transition. For the model, because it is to be benchmarked against a standard stock and flow model, the delay time for shipment is kept at exactly three days, with no variation – as is the case with equation (10) earlier. [A day is modelled as 480 minutes].

Number	Transition Name	Next Transition	Average Time	Standard Deviation
1	Wholesaler Start Order	2	20	2
2	Wholesaler Finish Order	3	10	0
3	Ship From Wholesaler	4	1440	0
4	Stock Arrives at Retailer	5	15	7
5	Retailer Start Order	6	20	10
6	Retailer Finish Order	7	10	0
7	Ship From Retailer	-	-	-
8	*Fire Retailer Stock Heuristic*	8	480	0
9	*Customer Demand Event*	8	480	0

Table 3: Transitions for the single-actor beer game

The places in the model are shown in table 4. These represent the different states, and some of them map directly onto the stocks presented in the earlier model. In cases where there are no direct mappings, this reflects the finer level of granularity that is present in the discrete-event model, where times are allocated for each processing step. However, it is not expected that these slight variations will significantly effect the overall behaviour of the model, because the reordering only occurs once at the end of each day.

Number	Name	Mapping to Stock and Flow Model
P1	Retailer Orders Queuing	No direct mapping
P2	Retailer Orders Being Processed	No direct mapping
P3	Retailer Orders Completed	No direct mapping
P4	Retailer Orders in Transit	Supply Line (5)
P5	Retailer Stock on Hand	Stock (2)
P6	Customer Orders	Order Queue (1)
P7	Customer Orders Being Processed	No direct mapping
P8	Customer Orders Completed	No direct mapping
P9	Total Customer Orders Delivered	No direct mapping

Table 4: Places for the single-actor beer game

Petri Net System Design

A purpose-built timed Petri net modelling system was developed, and the single actor beer game model constructed with this. The system is fully object-oriented[1], written in C#, and all results are stored in a relational database system, so that queries and statistics can be generated after each simulation run. The main components are:

- The Engine Sub-System, which contains an event list, calendar and the algorithm for executing all pending transitions in the system.
- The Model Sub-System, which defines key Petri net concepts such as places, transitions and tokens, and also defines the linkages between these different model components. For example, it allows places to be linked to transitions, and tokens to be added to places.
- The Transition Rules Sub-System, which contains the logic needed to deal with different kinds of transitions that arise in a modelled system. For example, there is a rule to deal with an exogenous event, a rule to model how an agent controls their stock levels – as in the beer game model, and rules to deal with more standard Petri net transitions.
- The Utilities Sub-System, which contains useful classes to generate random normal variables, and also classes that generate unique identification codes for events and tokens.

The relational database – implemented in SQLServer – contains interlinked tables, including:

- Event (EventID, Name, ClockTime, DayNumber, EventTime)
- BusinessObjectEvent(EventID, BusinessObjectID)
- BusinessObject(BusinessObjectID, Type)
- StateHistory(StateID, ClockTime, PlaceNumber, NumberTokens, StateTime)

[1] Over 40 independent classes, seven sub-systems and 10 relational database tables

These tables allow all events (which are in fact a record of a transition that has fired) to be linked to one or more business objects (for example, a customer order), so that a full analysis can later be performed on all the simulation data. Furthermore, the StateHistory table (table 5) allows snapshots of the state to be recorded, and this is needed so that calculations such as exponential average of orders can be carried out as part of the stock control heuristic. It also allows a daily record of state data to be maintained so that charts and reports can be constructed after the simulation has completed. The *Event*, *BusinessObjectsEvent*, and *BusinessObject* tables relate individual events back to business objects (where a business object can be an order or a resource). This enables full traceability and provides the necessary timed data in order to calculate overall throughput and utilisation metrics.

STATE HISTORY

ClockTime	Place	Tokens	Day	Time
0	5	0	11/01/2006	08:00:44
475	5	0	11/01/2006	15:55:50
955	5	0	12/01/2006	15:55:07
1435	5	0	13/01/2006	15:55:04
1915	5	0	16/01/2006	15:55:33

EVENTS

EventID	Name	ClockTime	DayNumber	Day	Time
EVENT.31LWIELY	Retailer Start Order	18448	39	06/03/2006	11:28:36
EVENT.XNN2800E	Retailer Finish Order	18461	39	06/03/2006	11:41:24
EVENT.AYNYL6YC	Ship From Retailer	18482	39	06/03/2006	12:02:30

BUSINESS OBJECT EVENTS

EventID	BusinessObjectID
EVENT.31LWIELY	CUST.ORDER.010RCMYH
EVENT.AYNYL6YC	CUST.ORDER.010RCMYH
EVENT.XNN2800E	CUST.ORDER.010RCMYH

BUSINESS OBJECT

BusinessObjectID	Type
CUST.ORDER.006R2YX0	Order
CUST.ORDER.00BBNLOJ	Order
CUST.ORDER.00ITPC7T	Order
CUST.ORDER.010RCMYH	Order
CUST.ORDER.014BTHKS	Order
CUST.ORDER.019NI3JG	Order

Table 5: Sample data from selected tables

Comparison of Approaches

In this section, the two approaches are compared. The first comparison summarises differences the model building and execution tasks. The second comparison shows the results to a number of experiments.

Comparison of Model Building and Execution

Building the two models resulted in the following observations:

1. The system dynamics model was easy to create, was able to accommodate all the decision rules as equations, and was straightforward and quick to run.
2. The Petri net model was 80% straightforward in terms of places, tokens, and transitions. However, two special transitions had to be coded directly: one to generate the customer demand, the other to generate the desired orders at the end of each simulated day. It seems unavoidable that some degree of computer coding is needed in order to overlay decision heuristics on a Petri net model. Also, a special purpose algorithm had to be written in order to calculate the expected orders value, which had been easily implemented using a negative feedback structure in Vensim (see Forrester 1961, pp 406-411 for detail on the relationships between exponential smoothing and information delays).
3. For the Petri net system, a special-purpose transition has to be written in order to record the state of all places at the end of each day. This would not be needed in a tool such as Vensim, Stella or Powersim, as the state variables are automatically recorded for each solution interval. Therefore, it was necessary to create, and populate, the table *State History* in the database.
4. For the Petri net system, it was necessary to create a relational database so that all results could be easily analysed, and that detailed workflows of individual orders through the system could be gathered. Of course, this level of detail on individual orders was unavailable in conventional system dynamics models.
5. The Petri net model took almost three minutes to run (generating over 160,000 transitions), while the Vensim model ran within an instant. While three minutes

seems within an acceptable range for analysis, if more advanced techniques such as optimisation were to be considered, then any individual simulation run that took more than a few seconds would become unattractive for detailed and extensive policy analysis. For example, genetic algorithm-based optimisations (Duggan 2005) often require up to 10,000 different simulation runs.

Experimental Results

The scope for experimentation with the two models is quite large, and is still ongoing, but some initial results are now presented based on the following runs:

- With supply line visibility:
  - Run both models with SAT (13) and SLAT (17) both equal to one.
  - Run both models with SAT and SLAT both equal to three.
- Without supply line visibility
  - Run both models with SAT (13) and SLAT (17) both equal to one.
  - Run both models with SAT and SLAT both equal to three.

Figure 6 summarises the results for the first run, where the supply line is taken into account and all adjustment times are equal to one. The data charted is the inventory level, and there are no significant differences except that inventory levels for the Petri net model drop to zero once during the "warm-up" phase of the model, and again around the time when the demand doubles from 100 to 200. With the adjustment times increased (figure 7), both models outputs are close, apart from the initial phase and when the step function is activated for customer demand. On observation, it seems that there is little difference between the two models, although this could be formally quantified using a sum of squares measure.

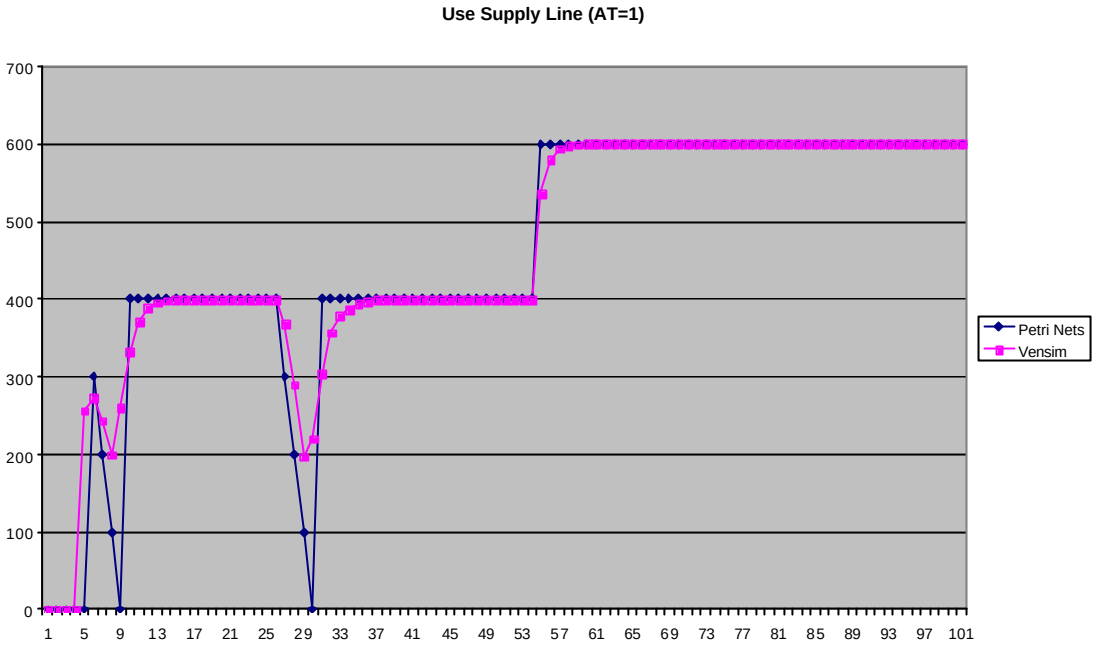

Figure 6: Plot of results for experiment 1

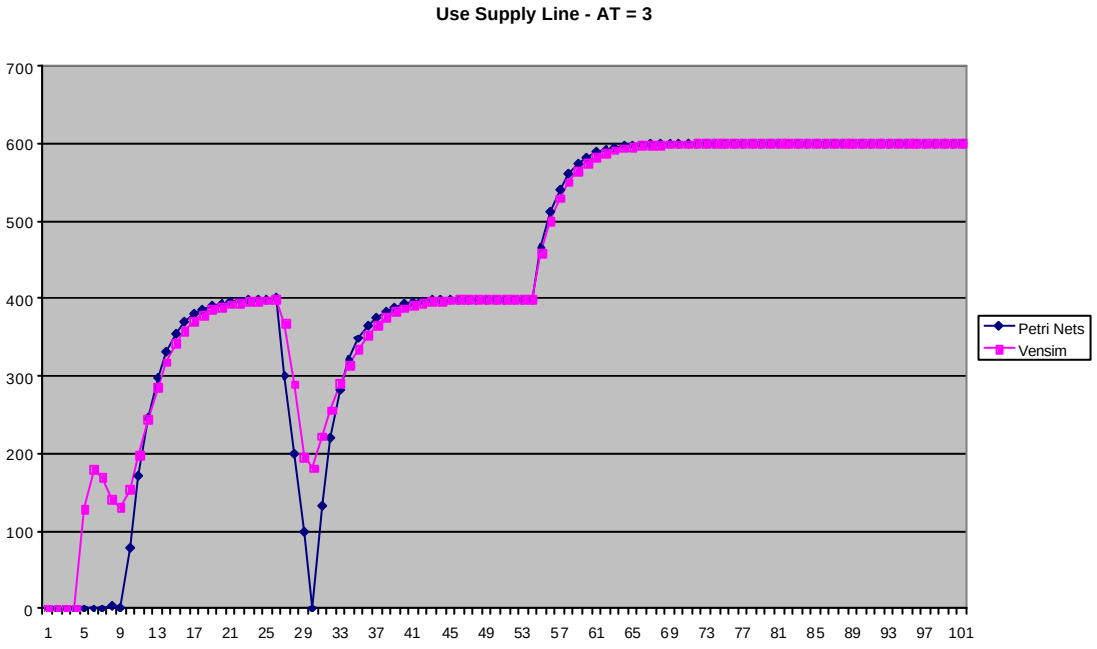

Figure 7: Plot of results for experiment 2

For the second set of experiments, the supply line visibility is deactivated. In the first case – figure 8, where the adjustment times are 1, while the behaviour from each model is similar (oscillation), they are slightly out of phase, and the peaks of the Petri net model are significantly higher. Further analysis is needed to explore the contributing factors for these differences.

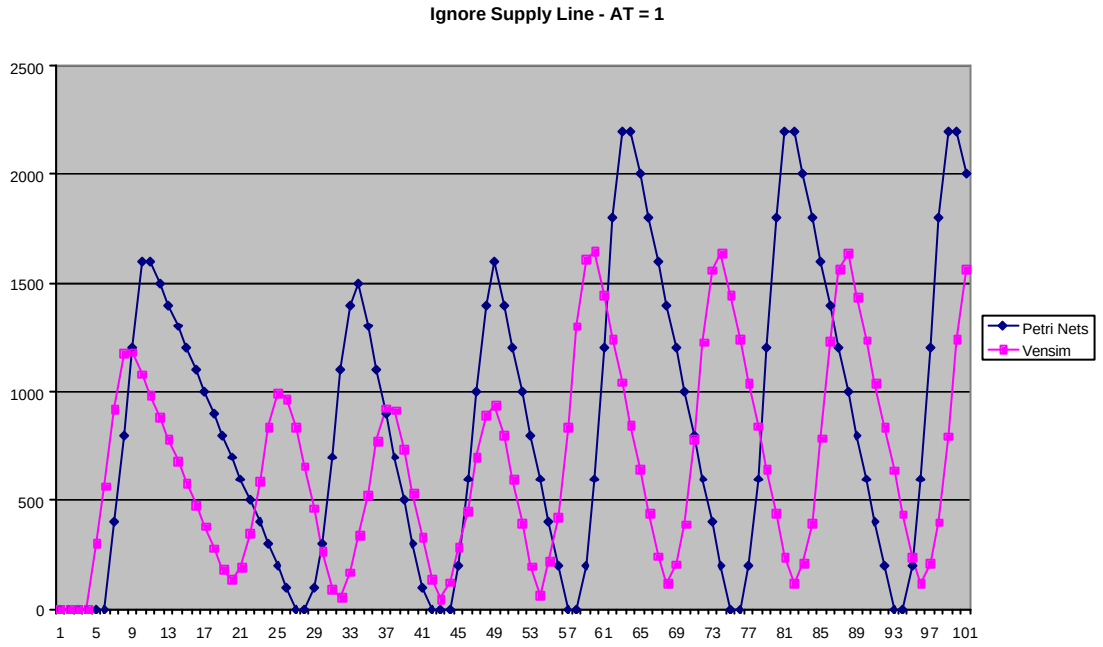

Figure 8: Plot of results for experiment 3

Finally, in figure 9, where the adjustment times are set at three, at an early stage of the simulation the peaks in the Petri net model are higher, although as time moves forward, the behaviour of both models converges towards the revised stock goal (600), and from time 61 onwards, the results are virtually indistinguishable.

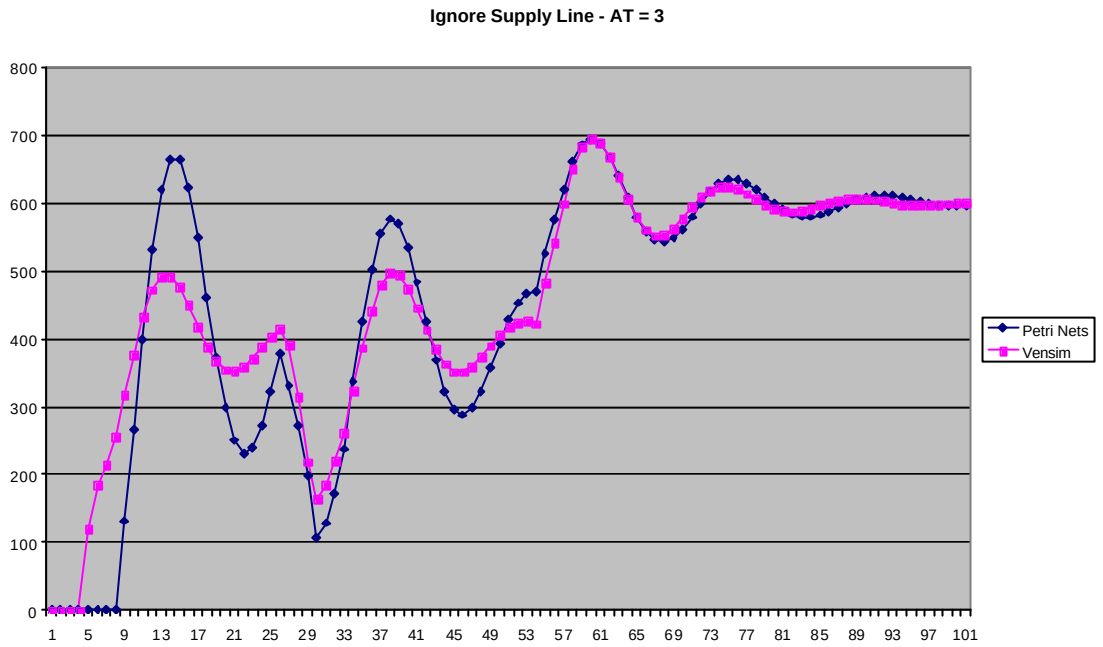

Figure 9: Plot of results for experiment 4

In conclusion, the closeness of model outputs varies depending on whether the supply line is visible, and also is dependent on the adjustment times used. From a dynamic behaviour viewpoint, the Petri net model replications the classic behaviours of oscillation and goal seeking behaviour, but further more detailed analysis is required to explain (1) the differences in peaks when the supply line is switched off and (2) why the initial stages of the models also vary.

Conclusions

The goal of this paper was to provide insights into how DES and SD of how these approaches can be used in a specific circumstance. In particular, it set about to explore whether Petri nets could replicate the classic behaviour of the standard system dynamics models. To a large degree, this has been successfully verified with the results, although further work is needed to explore smaller variations in the model, and this work could also include scaling up the model to the conventional four-actor game. Furthermore, it is hoped that this work can add to the wider discussion of where common ground can be explored between the theory and practice of discrete event simulation and system dynamics.

References

Banks, J, Carson, J.S., Nelson, B.L., Nicol, D.M. 1999. *Discrete-Event System Simulation.* Prentice Hall, Upper Saddle River, NJ.

Bauer, A, Bowden, R, Browne, J, Duggan, J., and Lyons G. *Shop Floor Control Systems: From Design to Implementation.* Chapman and Hall.

Brailsford, S.C. and Hilton, N.A. 2000. "A Comparison of Discrete Event Simulation and System Dynamics for Modelling Healthcare Systems." *Proceedings of ORAHS,* Glasgow Caledonian University, pp. 18-39.

Duggan, J. 2005. "Using Multiple Objective Optimisation to Generate Policy Insights for System Dynamics Models." *23rd International Conference of the Systems Dynamics Society*, Boston, July 2005.

Forrester, Jay W. 1961. *Industrial Dynamics*. Productivity Press, Portland, Oregon.

Lane, D.C. 2000. "You Just Don't Understand Me: Modes of Failure and Success in the Discourse between System Dynamics and Discrete Event Simulation." LSE OR Department Working Paper LSEOR 00-34, London School of Economics and Political Science.

Morecroft, J. and Robinson, S. 2005. "Explaining Puzzling Dynamics: Comparing the Use of System Dynamics and Discrete Event Simulation." *23rd International Conference of the Systems Dynamics Society*, Boston, July 2005.

Senge, P.M. 1990. *The Fifth Discipline: The Art & Practice of The Learning Organization.* Doubleday, New York.

Sterman, J.D. 1989. "Modeling managerial behaviour: Misperceptions of feedback in a dynamic decision making experiment." *Management Science*. 35 (3), pp 321-339.

Sterman, J.D. 2000. *Business Dynamics. Systems Thinking and Modeling for a Complex World*. McGraw Hill Higher Education.