RECENT ADVANCES IN LOGIC TABLES FOR REUSABLE DATABASE ENGINES

Colin James III
CEC Services, LLC
1613 Morning Dr
Loveland, Colorado 80538-4410 USA
(970) 622-0466, (970) 622-0177 (fax), cjames@cec-services.com

ABSTRACT

Report Accounts [RA] is a very fast reusable database engine for accounting arithmetic implemented in 10-tables and less than 25-lines of ANSI SQL code. RA has been generalized into abstract engines for process engineering and flexible manufacturing systems [FMS].

Recent advances are:

1. How to chain logic tables for multi-level design of complex systems.

2. How to map a database engine with multi-level logic tables into a Petri Net [PN].

3. How to recognize patterns in such PNs for reuse as frameworks in other systems.

4. How to recognize frameworks in Kanban, FMS, and product-form queueing networks [PFQN].

INTRODUCTION

This paper is divided into six parts: 1. Summary of previous work; 2. Analysis; 3. Design; and 4. Implementation.

1. SUMMARY OF PREVIOUS WORK

The previous work below is summarized from James (1998) and from recent results.

For Report Accounts [RA], the problem domain was determined through analysis from the requirements from which the design of the solution domain became obvious for implementation. Business Object Notation [BON] was used for analysis and design, and relational database management systems [RDBMS] technology was used for implementation. BON determined and isolated the components of attributes, objects, and classes. These in turn were mapped on a one to one basis directly into the respective columns, rows, and tables of RDBMS technology. The mapping process was intuitive, seamless, and reversible and quite unlike the current methodological tools for computer aided software engineering [CASE] which promised but could not deliver the same.

The specific requirements for RA were a database engine for accounting arithmetic to support:

1.1.1. Bookkeeping with user specified logic such as double- and triple-entry bookkeeping using credits [cr], debits [dr], and trebits [tr]

1.1.2. Reporting of all financial instruments

1.1.3. Accountability to trace back all transactions, also known as generally accepted accounting principles [GAP]

1.1.4. Portability to any major hardware / RDBMS platform

1.1.5. Maintainability in a ubiquitous ANSI computer language

1.1.6. Scalability of four billion potential concurrent users

1.1.7. Performance of not less than three seconds for the real time commit of a complete accounting transaction

The requirements were met or exceeded in these respective implementations of RDBMS technology:

1.2.1. N-entry bookkeeping is achieved through a logic table specified by the user where columns are transaction type numbers and rows are the account numbers on which respective switches are encoded with the type of arithmetic operation.

1.2.2. Reporting is achieved with the same logic table but with columns viewed in a separate numeric range as report type numbers, rows as account numbers, and switches encoded for the tab and row positions within a page and the type of reporting arithmetic.

1.2.3. Accountability is achieved in table design. The transaction table grows infinitely where truncations can not be altered per se but may be reversed through another transaction, all indexed to and from the accounts table.

1.2.4. Portability is achieved through RDBMS using ANSI Structured Query Language [SQL] which is a database access language as opposed to a procedural language. The SQL code to access the logic table uses the SUBSTR substring function which returns all row numbers specified by switches present in the column number of interest. This avoids reading each row in its entirety with the INSTR instring function. The logic table design and SUBSTR

function implementation effectively coerce SQL to perform the procedural processing inherent to accounting arithmetic. Hence non-procedural SQL achieves the portability not realized by procedural computer languages.

1.2.5. Maintainability is achieved with the design of only ten tables and implemented in less than 25-lines of ANSI SQL code.

1.2.6. Scalability is achieved by the choice of the RDBMS vendor. Currently IBM DB2 supports the maximum number of concurrent users.

1.2.7. Performance is achieved by table design and by the choice of the RDBMS vendor. RA is implemented in a minimum number of ten tables. The accounts table contains all account entries. A separate table for account balances contains one row respectively for the current running balance of each account number to ensure fast lookup. The relative performance of RA by vendor is currently as follows: IBM DB2 is 24-times faster than ORACLE; and CA OpenIngres is 6-times faster than ORACLE. Sybase and Informix were not tested due to extreme difficulties in obtaining those products with or without support.

2. ANALYSIS

This section describes how to stack or chain logic tables to perform multiple processes or operations on different levels.

2.1. RA uses a logic table of columns as transaction type numbers, of rows as account numbers, and of switches as the desired arithmetic operation of cr, dr, tr, or blank for none. The user specifies a transaction type number to access the logic table. That transaction type number is associated with a column, either directly by columnar position or indirectly by an index. That column is searched for any switches not set to blank. Those switches found then designate the particular rows and associated account numbers to which the transaction amount is applied as either cr, dr, or tr. Thus a single transaction selects multiple accounts.

2.2. We index the logic table in 2.1 above with a logic table for tasks where columns are task type numbers, rows are transaction type numbers, and switches are Boolean flags. Thus a single task selects multiple transactions which in turn select multiple accounts.

2.3. We also index the logic table in 2.2 above with a logic table of times where columns are time types, rows are task type numbers, and switches are Boolean flags. Thus a single time selects multiple tasks which in turn select multiple transactions and accounts.

2.4. We generalize the indexing of the logic tables in 2.1-2.3 above into abstract logic tables of level N. Thus a single operation at level N selects multiple operations at level N-1 which in turn select multiple operations at level N-2, at level N-3, …, and at Level N-(N-1) or Level 1.

The processes designated as columnar major in the respective logic tables above are times, tasks, and transactions. Other processes and operations may also be designated depending on the area of interest. For example in manufacturing, the columns in ascending order of level may be parts, inventories, assembly lines, shifts, work days, deadlines, projects, and systems. To abstract further, for example in computer science, the columns in ascending order of level may be neural networks, fuzzy logics, artificial intelligences, genetic algorithms, self-teaching systems, and self-replicating universes.

3. DESIGN

What follows from the analysis above is the necessity to capture the levels of chained logic tables into a graphical network hierarchy for discussion, manipulation, and simulation. To that end the Petri Net [PN] is an available tool.

The PN for RA and the logic table of 2.1 above is in Fig. 1.

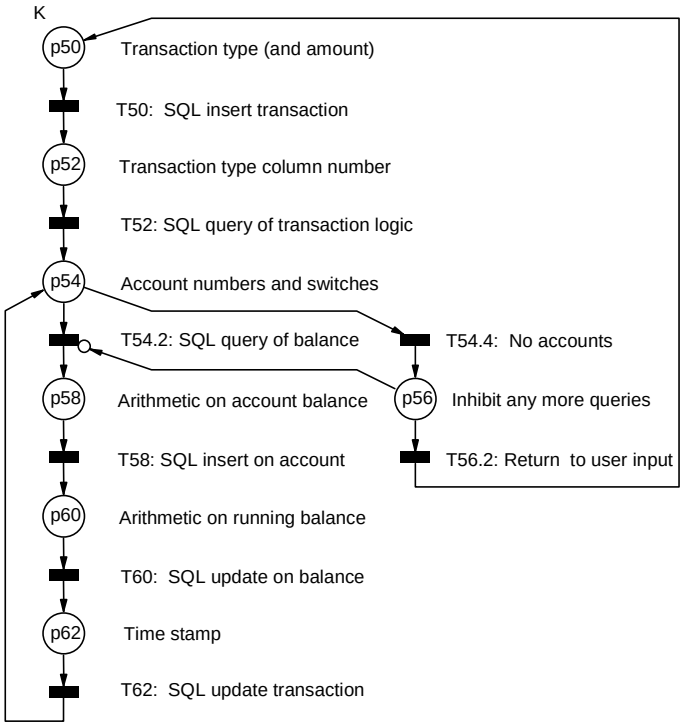

Fig 1. PN engine for one logic table

The mathematical definition of the PN in Fig. 1 is after Marsan (1995) below:

Definition. A PN model is an 8-tuple

$$M = \{P, T, I, O, H, PAR, PRED, MP\} \quad (3.1)$$

where

P is the set of seven places, P = {p50, p52, p54, p56, p58, p60, p62};

T is the set of transitions, $T \cap P = \varnothing$;

I, O, H : $T \rightarrow Bag(P)$, are the input, output and inhibition functions, respectively, where Bag(P) is the multiset of P;

PAR is a set of parameters, such as PAR = {K};

PRED is a set of predicates restricting parameter ranges, such as PRED = {K≥1};

MP : P → *IN* U PAR is the function that associates with each place either a natural number or a parameter ranging on the set of natural numbers where MP associates the parameter K with p50, and the value 0 with all other places and where *IN* is the set of natural numbers: {0,1,2,3, …};

Examples of the input, output, and inhibition functions are the following: I(t50) = {p50}, O(t50) = {p52}, H(t50) = ∅ and, for example, O(t54.2) = {p58} and H(t54.2) = {p56}.

The PNs for the logic tables described in 2.2 and 2.3 above are in Fig. 2 and Fig. 3, respectively, where the numeric levels of the chained logic tables are designated as dimensions.

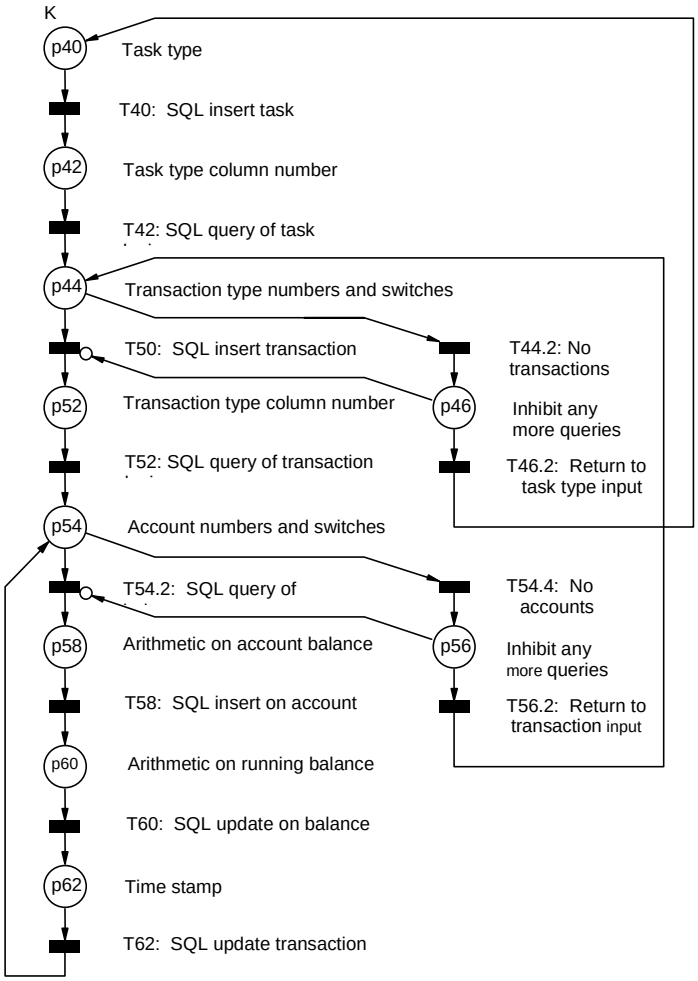

Fig. 2. PN for two logic tables (two dimensions)

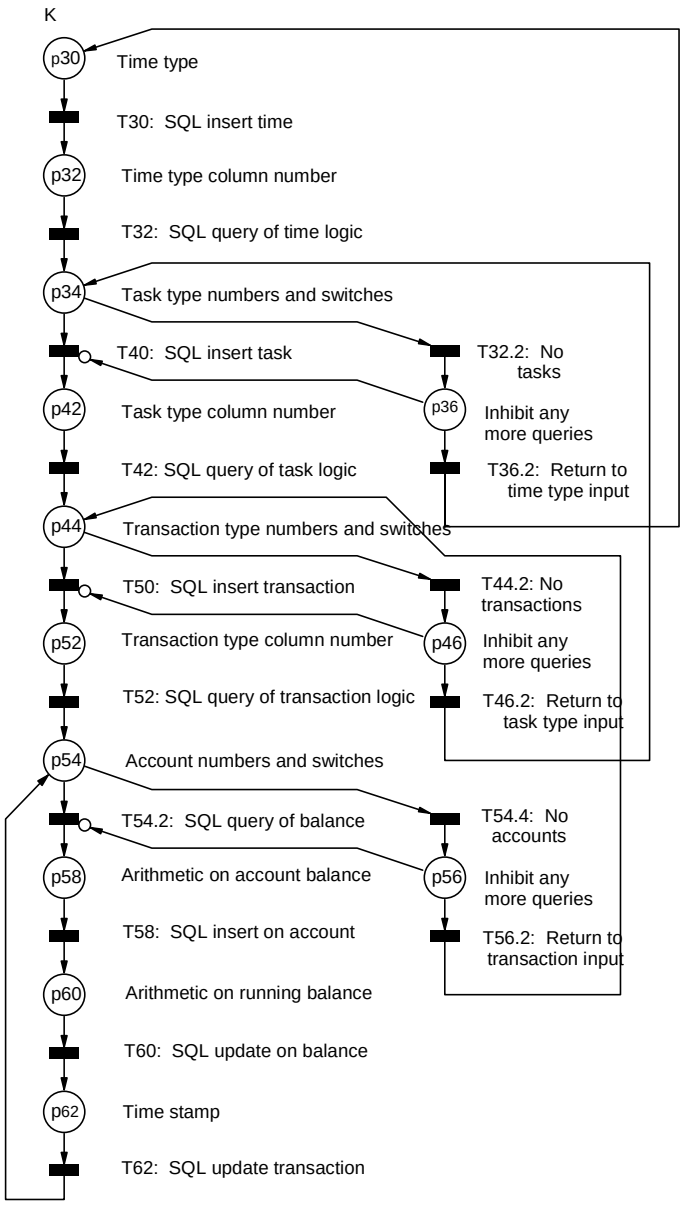

Fig. 3. PN for three logic tables (three dimensions)

4. IMPLEMENTATION

What follows from the PN design of chained logic table systems is how to abstract them into patterns and frameworks for generic reuse. To that end, the multiple levels of the logic tables described in 2.3 above and in Fig. 3 are abstracted into the PN pattern in Fig. 4 for three dimensions. The multiple levels of the logic tables described in 2.2 above and in Fig. 2 are abstracted into the PN pattern in Fig. 5 for two dimensions. The dotted area in Fig. 5 is shown in Fig. 6 as the abstract framework for logic table systems for N dimensions/levels.

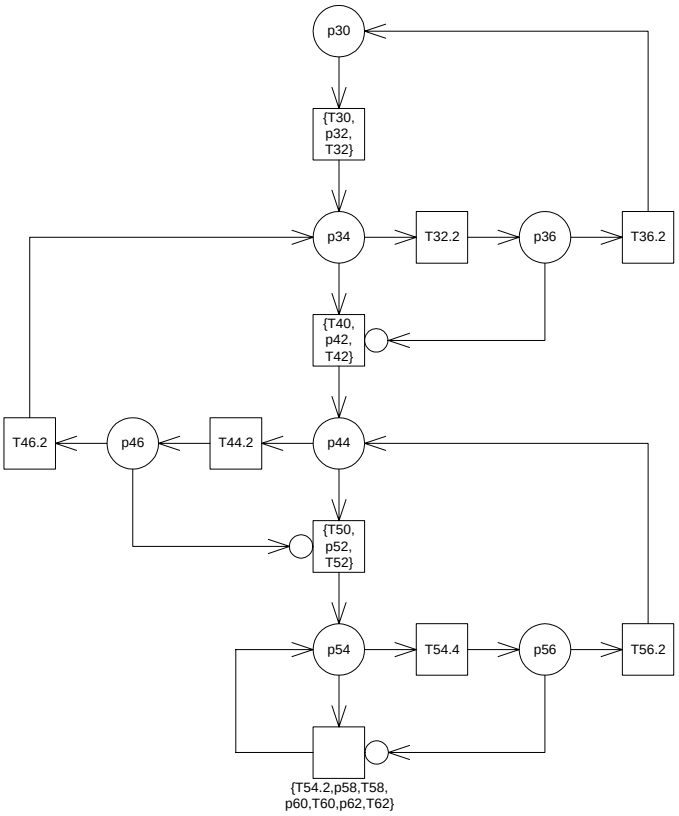

Fig. 4. Abstract PN pattern for three logic tables (three dimensions) from Fig. 3

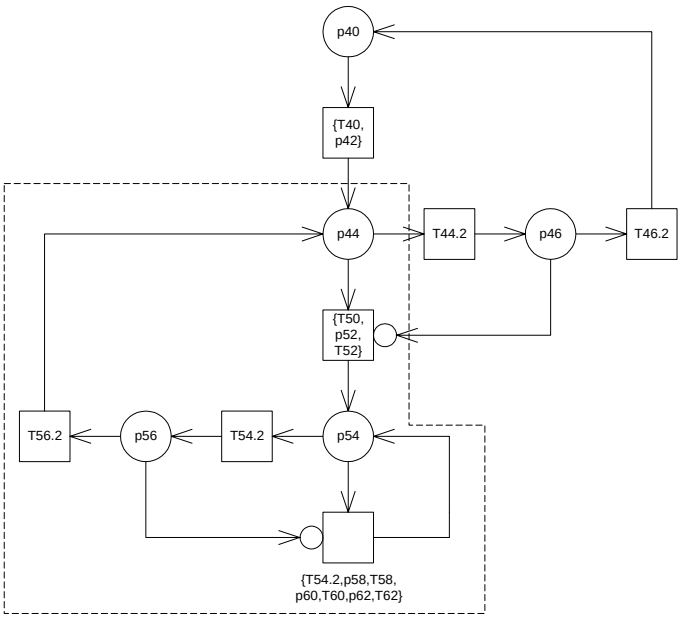

Fig. 5. Abstract PN pattern for two logic tables (two dimensions) from Fig. 2

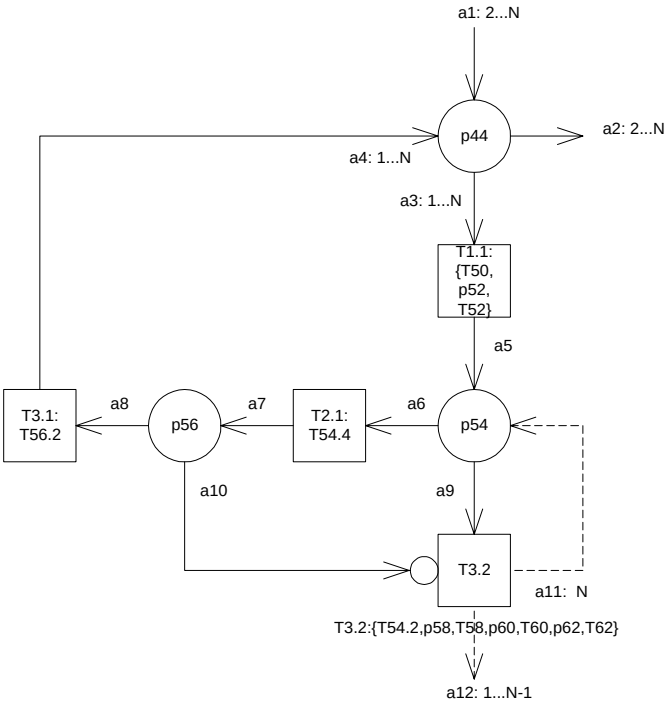

Fig. 6. Abstract PN framework for logic tables (N dimensions) from the dotted area in Fig. 5

In Fig. 6, the directed arcs of a1:2…N and a2:2…N are labeled to show where the pattern of logic table system levels 2 through N connect to place p1. The directed arcs of a3:1…N and a4:1…N show that levels 1 through N of the pattern are bounded by {p44, T3.1, T3.2, p54}. The place p56 is a test and the equivalent condition in the SQL query where more rows remain to be processed as arc a8 or no rows remain as arc a10. The dotted and directed arc of a12:1…N-1 shows the pattern where levels 1 through N-1 connect. The dotted and directed are of a11:N shows the pattern associated with level N.

A search of the literature for potential similarities in other PN patterns with inhibited arcs found Marsan (1995), Balbo (1985), and Couvreur (1994) which are described below, respectively, in Fig. 7 - Fig. 17, Fig. 18 - Fig. 30, and Fig. 31 - Figs. 33. In the area of manufacturing, a Kanban cell is shown in Fig. 7-Fig. 11, and a flexible manufacturing system [FMS] is shown in Fig. 12-Fig. 17.

A Kanban cell is the production part of a linear pull system based on the just in time [JIT] method of control to minimize the size and change of inventory.

Figure 7 is a Kanban cell after Marsan (1995, Fig. 88) which can fail with a failure subnet {m6, t6, m7, t7, and inhibited arc to t2} and with an idle subnet {t3, m5, t2}.

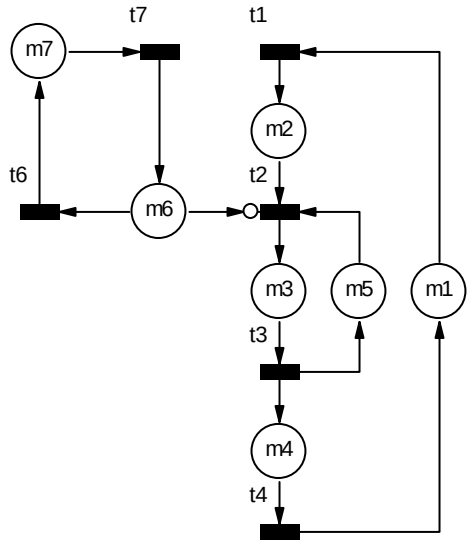

Fig. 7. Model of a Kanban cell with failure and idle subnets after Marsan (1995, Fig. 88)

Figure 8 redraws Fig. 7 with the transition t2 and the failure subnet at the bottom.

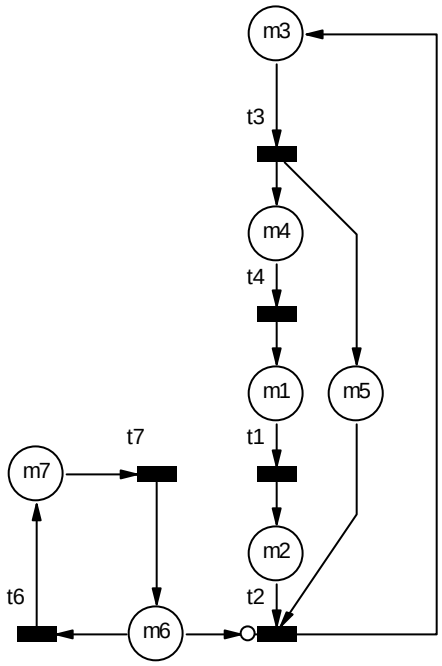

Fig. 8. Redrawn Kanban cell, identical to Fig. 7

Figure 9 expands the size of the failure subnet and reduces the main net {m3, t2} while retaining the idle subnet now {m5, t2, m5}.

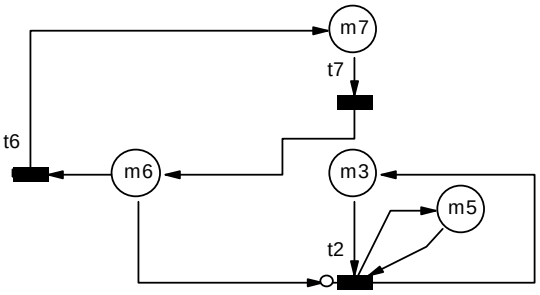

Fig. 9. Reduced Kanban cell from Fig. 8

Figure 10 suppresses the idle subnet and looks similar to the abstract PN framework for logic tables in Fig. 6 on the following basis. In Fig. 10 the construct {m7, t7, m6} passes by place m3 in contrast to the equivalent construct in Fig. 6 of {p44, T1.1, p54, T2.1} which does not pass by p54, the equivalent of place m6. Figure 10 is also identical to the elementary net system with inhibitor arc [ENI] of Janicki (1995, Fig. 4(a)).

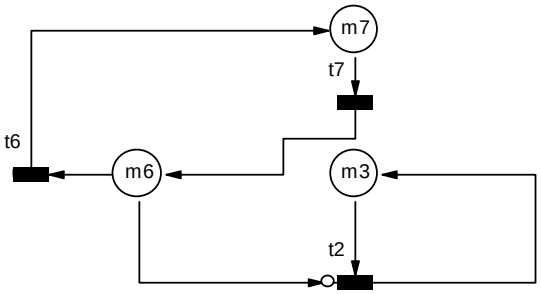

Fig. 10. Kanban cell of Fig. 9 with idle subnet suppressed and similar to the abstract PN framework for logic tables in Fig. 6

Figure 11 transforms Fig. 10 by connecting the failure subnet {m7, t7, m6, t6} into the main net {m3, t2, m3} at place m3 by forcing transition t7 into place m3. The reason for this follows. To simulate Fig. 10 correctly requires placing a token in place m7 and in place m3, that is, two tokens are required. However, it is possible to make a PN which simulates correctly with only one token a Kanban cell which can fail. Such is Fig. 11 where the failure subnet is connected directly into the main net at m3, the only place in the main net. Hence Fig. 11 is identical to the abstract PN framework for logic tables in Fig. 6.

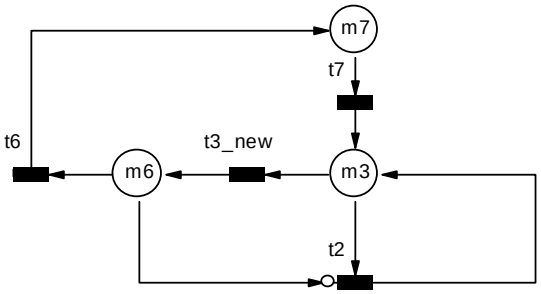

Fig. 11. Kanban cell of Fig. 10 with failure subnet connected into the main net and identical to the abstract PN framework for logic tables in Fig. 6

FMS is a push production system using pallets to load incomplete parts and to unload completed parts by continuous transportation such as conveyer or by automatic guided vehicle [AGV].

Figure 12 is an FMS with AGV transport as a generalized stochastic Petri net [GSPN] after Marsan (1995, Fig. 99). There are four inhibited arcs between places {m5, m8}, {m6, m21}, {m28, m21}, {m29, m8}. There are two idle subnets with places {m17, m18, m17} and {m17, m23, m17}.

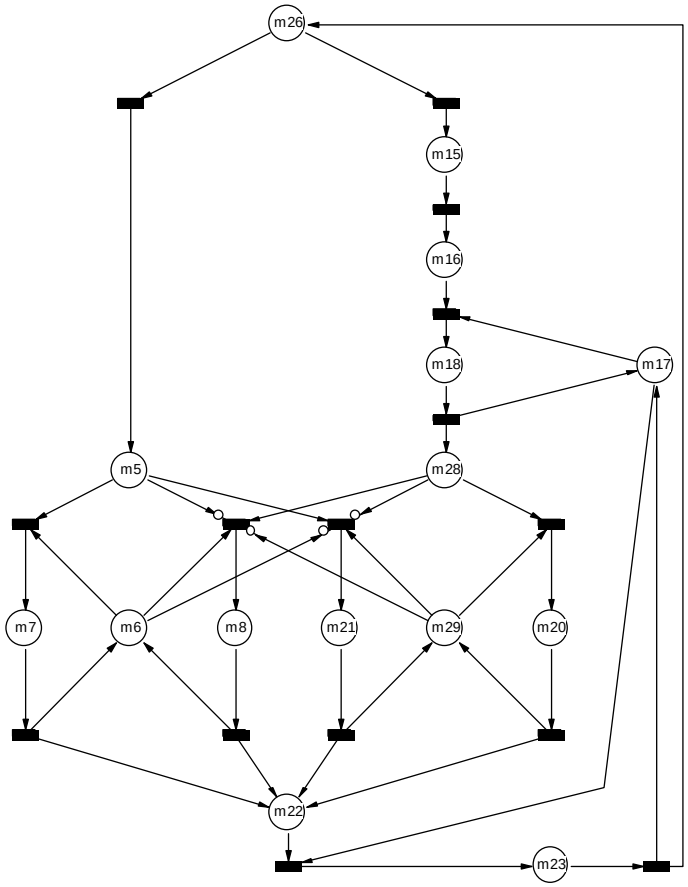

Fig. 12. GSPN model of a FMS with AGV transportation system after Marsan (1995, Fig. 99)

Figure 13 reduces Fig. 12 and moves to the bottom the place m21 of the two inhibited arcs and places {m6, m21}and {m28, m21}.

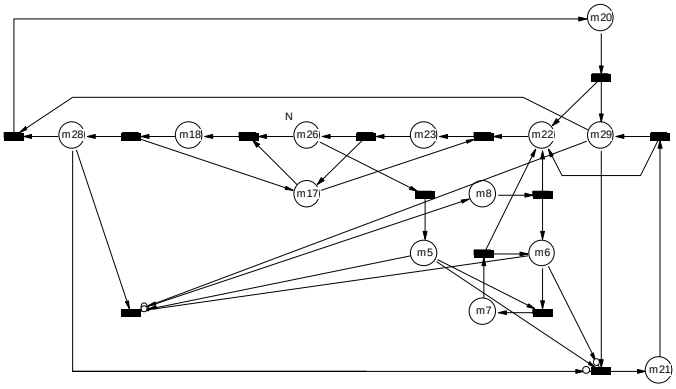

Fig. 13. Reduced GSPN model from Fig. 12

Figure 14 reduces Fig. 13 by removing the idle subnets and is identical to the abstract PN framework for logic tables in Fig. 6.

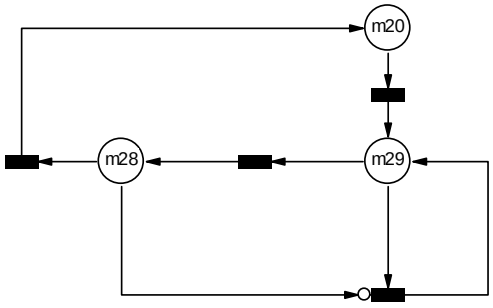

Fig. 14. Reduced GSPN model from Fig. 13 and identical to the abstract PN framework for logic tables in Fig. 6

Figure 15 reduces Fig. 13 and moves to the bottom the place m8 of the two inhibited arcs and places {m29, m8} and {m5, m8}.

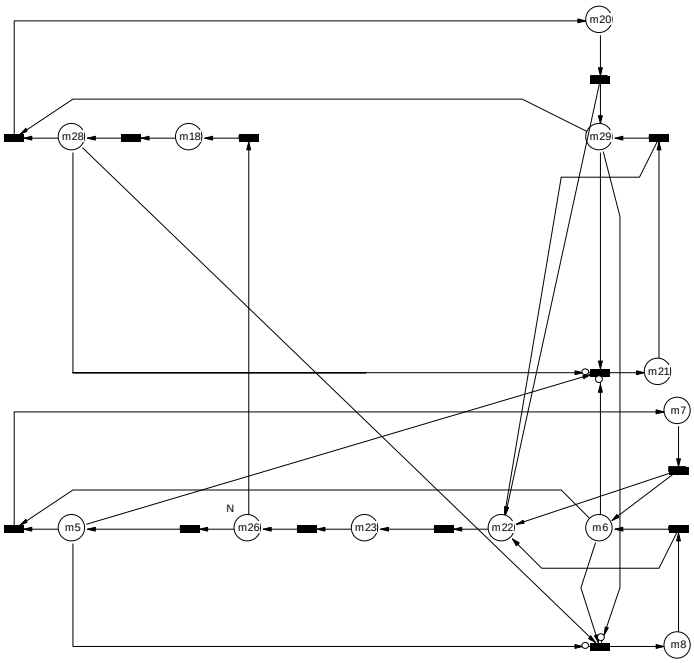

Fig. 15. Reduced GSPN model from Fig. 13

Figure 16 reduces Fig. 15 by suppressing the two inhibited arcs associated with place m21, namely {m6, m21} and {m28, m21}.

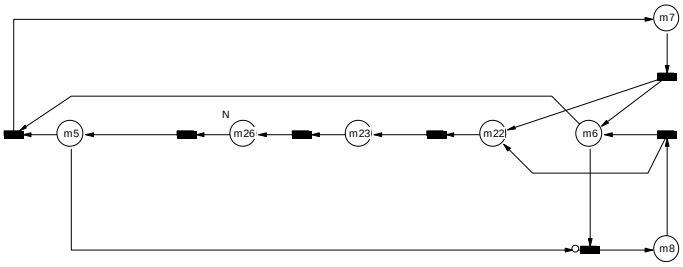

Fig. 16. Reduced GSPN model from Fig. 15

Figure 17 reduces Fig. 16 and is identical to Fig. 14 and to the abstract PN framework for logic tables in Fig. 6.

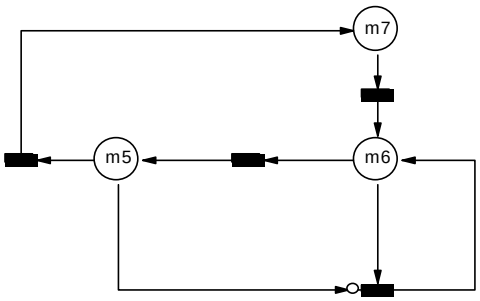

Fig. 17. Reduced GSPN model from Fig. 16 and identical to the abstract PN framework for logic tables in Fig. 6

In the area of software analysis, product-form queueing networks [PFQNs] provide models for analyzing performance where the numerical solution process, while although completely automated, becomes clearer with the help of GSPNs. A PFQN compact model is in Fig. 18-Fig. 23, and a PFQN lower-bound model is in Fig. 24-Fig. 30.

Figure 18 is a GSPN model of a PFQN compact model after Balbo (1985, Fig. 6).

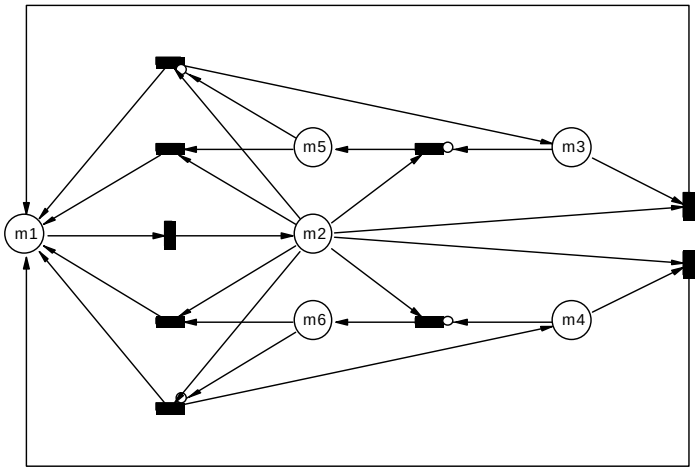

Fig. 18. GSPN model of a PFQN compact model after Balbo (1985, Fig. 6)

Figure 19 is the subnet {m1, m2, m4, m6, m1} on the lower half of Fig. 18.

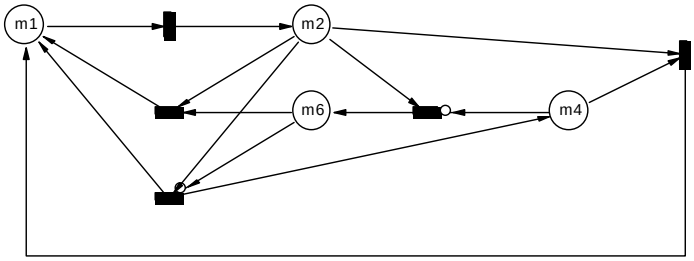

Fig. 19. Lower half subnet of the GSPN model from Fig. 18

Figure 20 redraws Fig. 19 using place m6 as the inhibited arc of interest.

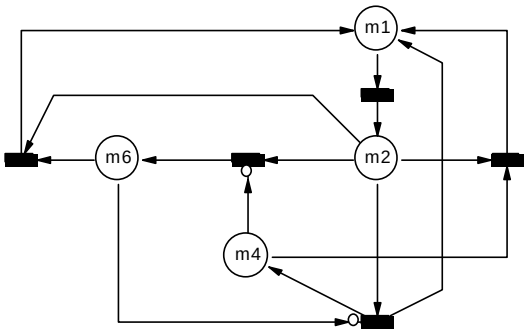

Fig. 20. Redrawn GSPN model from Fig. 19

Figure 21 reduces Fig. 20 and is similar to the abstract PN framework for logic tables in Fig. 6. The difference is that here the return loop is {m2, m1} rather than the equivalent {m2, m2) or {p54, T3.2} in Fig. 6. This difference is described in terms of the SQL code in Fig. 3 as abstracted in Fig. 6. Fig. 21 processes one account number at place m2 (or p54 in Fig.3 and Fig. 6), but then instead of checking for more account numbers to process in m2 goes back to get another transaction at place m1 (or p44).

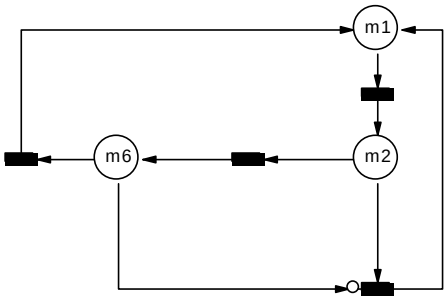

Fig. 21. Reduced GSPN model from Fig. 20 and similar to the abstract PN framework for logic tables in Fig. 6

Figure 22 transforms Fig. 21 by moving the return loop {m2, m1} into {m2, m2} because there is no branching between places m1 and m2. In Fig. 19, this changes the directed arcs {m2, m1} into the directed arcs {m2, m2}. Hence Fig. 22 is identical to the abstract PN framework for logic tables in Fig. 6.

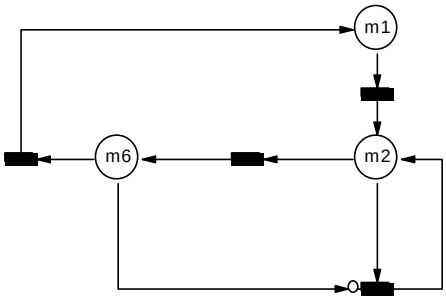

Fig. 22. Transformed GSPN model from Fig. 21 and identical to the abstract PN framework for logic tables in Fig. 6

Figure 23 redraws Fig. 19 using place m4 as the inhibited arc of interest (rather than m6 as in Fig. 20).

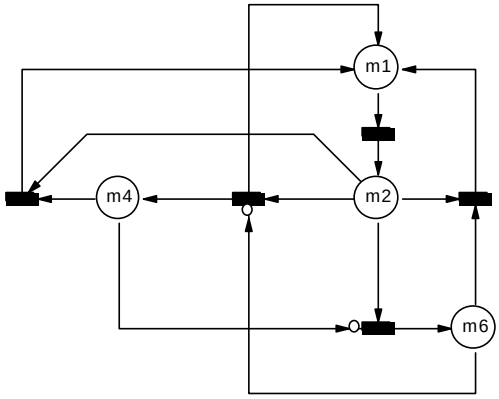

Fig. 23. Another redrawn GSPN model from Fig. 19

Figure 24 reduces Fig. 23 and is similar to the abstract PN framework for logic tables in Fig. 6.

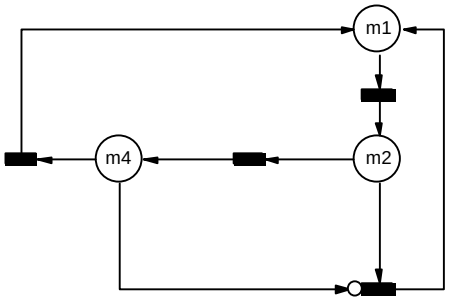

Fig. 24. Reduced GSPN model from Fig. 23 and similar to the abstract PN framework for logic tables in Fig. 6

Figure 25 transforms Fig. 24 by moving the return loop {m2, m1} into {m2, m2} because there is no branching between places m1 and m2. Hence Fig. 25 is identical to the abstract PN framework for logic tables in Fig. 6.

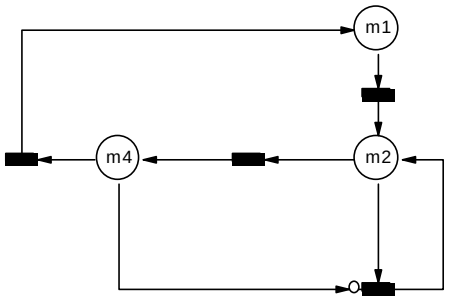

Fig. 25. Transformed GSPN model from Fig. 24 and identical to the abstract PN framework for logic tables in Fig. 6

Figure 26 is a GSPN model of PFQN lower-bound model after Balbo (1985, Fig. 7).

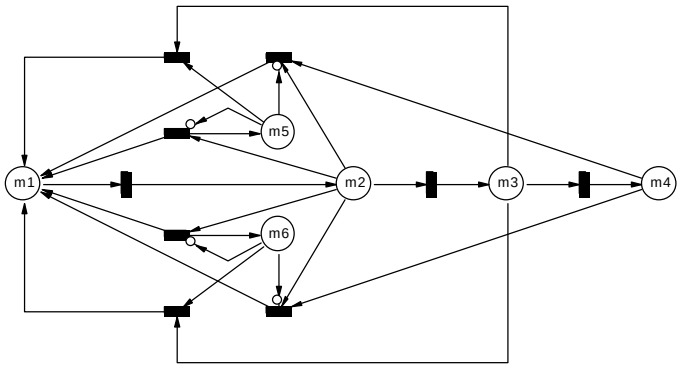

Fig. 26. GSPN model of a PFQN lower-bound model after Balbo (1985, Fig. 7)

Figure 27 is the subnet {m6, m1, m2, m3, m4, m1} in the lower half of Fig. 26.

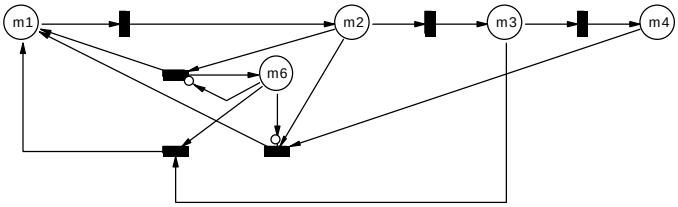

Fig. 27. Lower half subnet of the GSPN model from Fig. 26

Figure 28 redraws Fig. 27 using place m6 as the inhibited arc of interest.

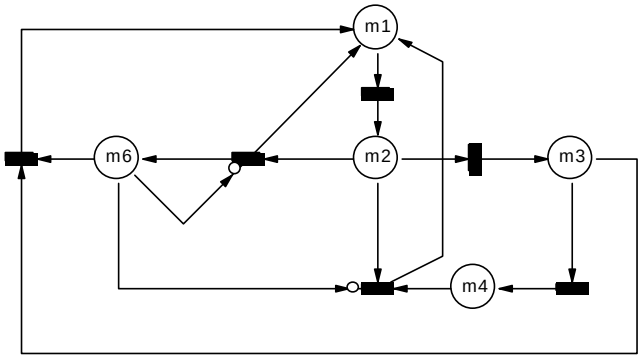

Fig. 28. Redrawn GSPN model from Fig. 27

Figure 29 reduces Fig. 28 and is similar to the abstract PN framework for logic tables in Fig. 6.

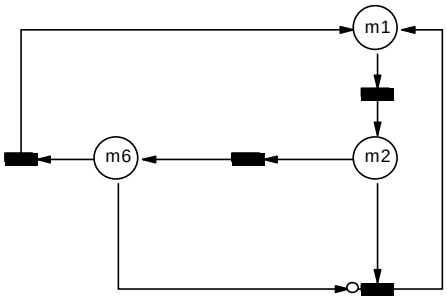

Fig. 29. Reduced GSPN model from Fig. 28 and similar to the abstract PN framework for logic tables in Fig. 6

Figure 30 transforms Fig. 29 by moving the return loop {m2, m1} in Fig. 29 into {m2, m2} in Fig. 30 because there is no branching between places m1 and m2. In Fig. 27, this changes the directed arcs {m2, m1} into the directed arcs {m2, m2}. Hence Fig. 30 is identical to the abstract PN framework for logic tables in Fig. 6.

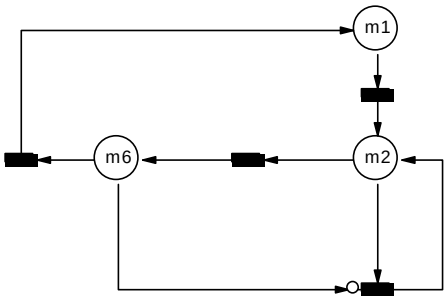

Fig. 30. Transformed GSPN model from Fig. 29 and identical to the abstract PN for logic tables in Fig. 6

In the area of validating models with invariants, the classic example of a communications protocol is in Fig. 31 - Fig. 33.

Figure 31 is an alternate bit protocol model after Couvreur (1994, Fig. 7).

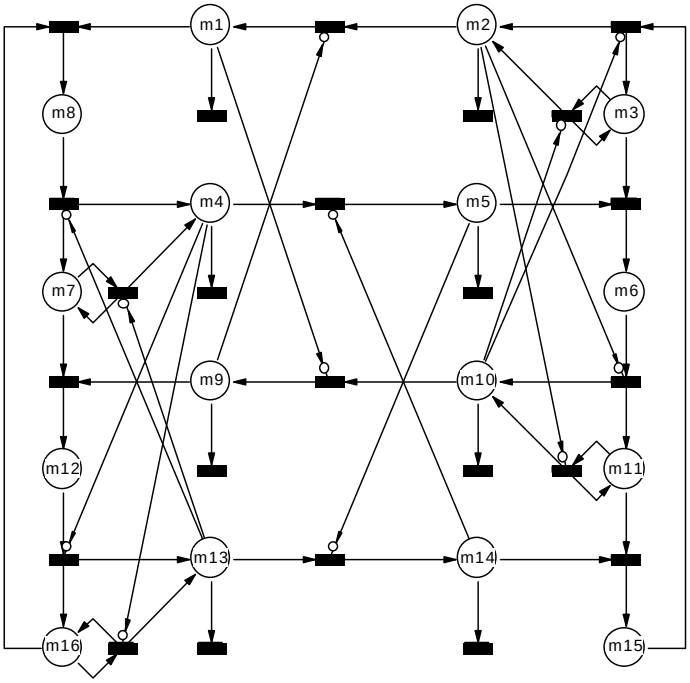

Fig. 31. Alternate bit protocol model after Couvreur (1994, Fig. 7)

Figure 32 reduces the alternate bit protocol model from Fig. 31 and is similar to the abstract PN framework for logic tables in Fig. 6.

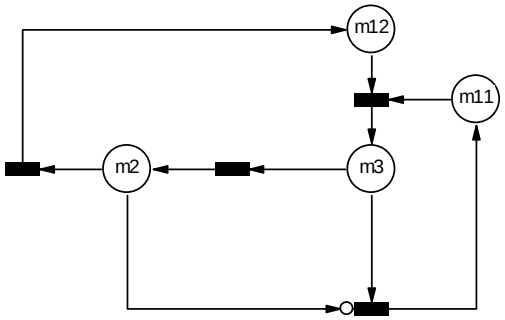

Fig. 32. Reduced alternate bit protocol model from Fig. 31 and not identical to the abstract PN framework for logic tables in Fig. 6

Figure 33 transforms the alternate bit protocol model from Fig. 32 by moving the return loop { m3, m11} in Fig. 32 into { m3, m3} in Fig. 33 because there is no branching between places m3 and m11. Hence Fig. 33 is identical to the abstract PN framework for logic tables in Fig. 6.

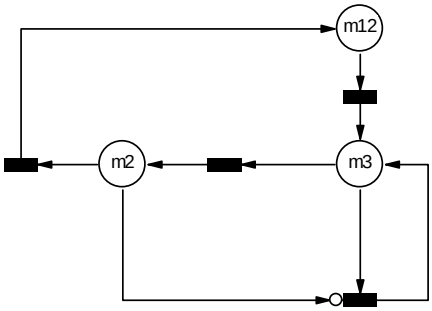

Fig. 33. Transformed alternate bit protocol model from Fig. 32 and identical to the abstract PN framework for logic tables in Fig. 6

CONCLUSION

This paper shows the exact processes of: how logic tables are chained to implement complex accounting and manufacturing systems; how to map such systems into PNs; how a common pattern was discovered which lead to a framework for reuse; and how that framework was further observed in existing examples of Kanban, FMS, and PFQN in the literature.

Janssens (1998) critiques Li (1994) for not producing a PN class due to limitations in modeling procedural knowledge and which avoid workflows. This paper addresses those objections by solving the problems of modeling workflows and the resulting impedance mismatch of procedural knowledge. The workflow model is reduced to an abstract PN framework which may be chained and represents the implementation of the procedural knowledge in its form of logic tables which also may be chained. Hence the abstract PN framework of Fig. 6 is proposed as unique to model both workflow and procedural knowledge at the same time. This new framework is one such study to "realize systematic reuse in the workflow field" as suggested by Janssens (1998). What follows is that a taxonomy is needed for the classification and codification of patterns and frameworks in PNs. Once a corpus of such components is assembled then some reuse may be possible.

A lesson learned is that object technology helps to define the problem domain by solving the analysis and design problem of representing levels of related transaction processing, but that only RDBMS technology can implement the solution domain in chained logic tables. The mapping from object technology is on a one to one basis directly into RDBMS technology. Hence there is no impedance mismatch and no need for a fuzzy, object-relational layer as a wrapper to abstract the relational database sufficiently for an object database to access it. Therefore it is now possible to avoid object-relational layers, such as those in the guise of middleware, and to avoid libraries of components, such as those in the guise of reusable objects and classes in procedural languages, through the use of complex logic tables in relational database engines. By extension, object technology has not kept its promise for programming by contract (to avoid errors as early as possible through invariants) or for reuse (to avoid reinvention through portable, generic components). The only place where object technology has helped is to generalize and abstract problem domains to the point where solution domains become obvious. On the other hand, RDBMS technology has enabled programming by contract (through a perfect mathematical data model) and realized reuse (through SQL as the simplest, most portable, and most ubiquitous ANSI language).

Another lesson learned is that the production of this paper underscores the dire need for better quality PN tools which are more readily available, less expensive than, and not bound to the Un*x / Sun environments.

ACKNOWLEDGMENTS

Thanks are due for helpful comments to: Professor Gurdeep Singh Hura, Department of Computer Science and Engineering, University of Idaho at Idaho Falls, Idaho Falls, ID; Professor Gerrit Janssens, Information Systems & Operations and Logistics Management, University of Antwerp - RUCA, Antwerp, Belgium; and Professor Houshang Masudi, Department of Mechanical Engineering, Prairie View A&M University, Prairie View, TX.

REFERENCES

Balbo, G., Bruell, S.C., and Ghanta, S., 1985, "Combining Queueing Network and Generalized Stochastic Petri Net Models for the Analysis of A Software Blocking Phenomenon", International Workshop on Timed Petri Nets, Torino, Italy, pp. 208ff, IEEE Computer Society Press, New York, New York.

Couvreur, J.M., Paviot-Adet, E., 1994, "New Structural Invariants for Petri Net Analysis", Application and Theory of Petri Nets, Proceedings of the 15th International Conference, Zaragoza, Spain, pp. 199-218, Springer-Verlag, Berlin, Germany.

James, C., 1998, "A Reusable Database Engine for Accounting Arithmetic", Proceedings of The Third Biennial World Conference on Integrated Design & Process Technology, Vol. 2, pp. 25-30, Berlin, Germany.

Janicki, R., Koutny, M., 1995, "Semantics of Inhibitor Nets", Information and Computation, Vol. 123, No. 1, pp. 1-16.

Janssens, G.K., Verelst, J., Weyn, B., 1998, "Reuse-oriented Workflow Modelling with Petri Nets", Workflow Management:Net-based concepts, models, techniques, and tools, 19 th International Conference on Application and Theory of Petri Nets, Lisboa, Portugal, pp. 40-59.

Li, J., Ang, J.S.K., Tong, X., Tueni, M., 1994, "AMS: A Declarative Formalism for Hierarchical Representation of Procedural Knowledge", IEEE Transactions on Knowledge and Data Engineering, Vol. 6, No. 4, pp. 639-643.

Marsan, M.A., Balgo, G., Conte, G., Donatelli, S., and Franseschinis, G., 1995, Modelling with Generalized Stochastic Petri Nets, John Wiley & Sons Ltd, West Sussex, England.