Davidrajuh, R. (2013) Distributed workflow based approach for eliminating redundancy in virtual enterprising. *The Journal of Sumpercomputing* 63(1), pp. 107-125

Link to published article:

DOI: 10.1007/s11227-010-0544-6

(Access to content may be restricted)

Distributed workflow based approach for eliminating redundancy in virtual enterprising

Reggie Davidrajuh

Abstract The existence of redundancy is a serious problem in virtual enterprise in which a number of collaborating enterprises join together to manufacture and sell a class of product for a time-limited period. This paper proposes a new approach for detection and elimination of redundancy in virtual enterprises; the proposed approach is based on workflow and uses a Petri net for modeling and simulation of workflows. This paper also presents a working example as a proof of concept.

Keywords petri net · workflow · redundancy · virtual enterprising · GPenSIM

1 Introduction

The research problem discussed in this paper is the existence of redundancy in virtual enterprises. Virtual enterprise means a nucleus enterprise (or main assembler) joins with a number of collaborating enterprises (supply and distribution enterprises, transporting agents), to manufacture and sell a class of product with the characteristics such as qualitative, agility, and leanness, to achieve maximum customer satisfaction [5, 6]. When market requirements are changed, a new class of products or an improved version of the product should be turned out to meet the new market requirements. In this case, the nucleus enterprise may seek for a new combination of collaborating enterprises that are more suitable to manufacture the new class of products; thus the main aspect of virtual enterprise is dynamic logic of organization and reorganization of collaboration [1].

Formation of a virtual enterprise is rather quick, as market opportunities for which the virtual enterprise is being formed are time-limited. Due to the time constraint and due to the independency of collaborating enterprises, when forming a virtual enterprise, collaborating enterprises are mainly concerned about division of labor and responsibilities for production, distribution, and sales of products; there is not much time left for any one of the collaborating enterprises to see what and how the other enterprise is going to perform its operations. Hence, there will be redundancy as two or more enterprises may be unknowingly doing the same operations, job, or functions.

2 Related works and our new approach

In this paper, we propose a new approach for eliminating redundancy in virtual enterprising. Before the new approach is presented, in order to better understand how this paper advances the state of the art, a systematic description of the related work is also given.

2.1 Related works

In the last decade, workflows have been studied intensively in the context of geographically distributed computing systems (Grids) [7]. In the context of Grid, also the concept of virtual organization (VO) was introduced [8]. Especially, in the context of e-Science, scientific workflows have been proved useful for collaboration of scientists from various organizations [10]. Many systems for workflow specification, planning, optimization, management have been proposed [18, 19]. Researchers have also proposed to use UML for high-level specification of workflows [2]. This paper proposes an approach based on distributed workflow.

As explained in the next subsection, this approach has roots in an approach proposed by van der Aalst and Weske [17]; however, the approach proposed by van der Aalst and Weske is not a distributed approach. In order to allow distributed workflow simulation, this paper also makes use of new tool known as GPenSIM [11].

2.2 Our new approach

Our approach is an iterative approach in which simulations are done iteratively until simulations results converge resulting in an optimal design. Our approach consists of four stages (Fig. 1):

- Stage-1: Starting with a crude global workflow: The collaborating enterprises agree on a virtual-enterprise wide global workflow; the global workflow clearly identifies main functional blocks (or domains, e.g. manufacturing, sales, suppliers, distributors, etc.) of the virtual enterprise and also the interfaces between the functional blocks. The global workflow serves as an outline (or contract) between these collaborative enterprises. Though the global workflow clearly identifies both functional blocks and the interfaces between them, the functional blocks are crude and lack details.
- Stage-2: Partitioning the global workflow into workflow modules: Based on the clearly identified functional blocks, the collaborating enterprises (or the dominating nucleus enterprise) may divide the global workflow into workflow modules.

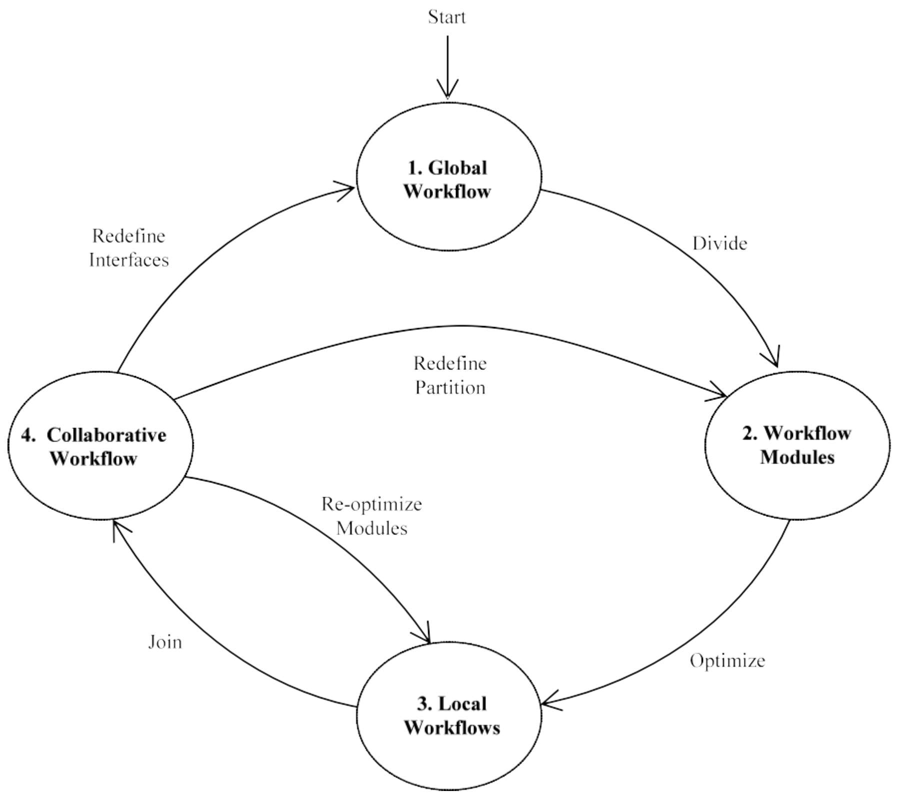


Fig. 1 Workflow based approach for eliminating redundancy in virtual enterprising

The collaborating enterprises will be given responsibility to realize (add details) one or more of the workflow modules; thus, collaborating enterprises get their own local workflows.
Stages 1 and 2 are based on a methodology proposed by [17].

- Stage-3: Optimizing local workflow modules individually (locally): The collaborating enterprises work on the workflow modules that are allocated to them. Firstly, they add details to the workflow modules so as to reflect the business processes. Then they optimize the workflow after finding and eliminating performance problems such as bottlenecks, redundancies, deadlocks, etc. Though the enterprises are free to develop their own workflow modules as they wish, they cannot change the already agreed interfaces between the modules.
- Stage-4: Global optimization: The individual locally optimized workflow modules are joined together to form the collaborative workflow. Simulations are done on the collaborative workflow to see whether it satisfies the overall targets of the virtual enterprise. Simulations may reveal two types of performance problems: global and local.

Global problems (such as overall delays, high product costs, etc.) affect the whole virtual enterprise; redesigning the initial global workflow with emphasis on redefining the interfaces between the functional blocks, or redividing the global workflow into different set of functional blocks may help solve global problems. Local problems affect individual enterprises only. Local problems are solved by redesigning the local workflow modules.

3 Technology and tools: Petri nets and GPenSIM

To make this paper self-contained, this section introduces the technologies and tools used in this paper namely Petri nets, and GPenSIM.

3.1 Petri nets

This section will give a brief introduction to Petri nets. For further details, interested readers are referred to [3]. Carl Adam Petri invented Petri nets in 1962, as part of his dissertation titled “Kommunikation mit Automaten” at the University of Bonn. This work significantly advanced the fields of parallel and distributed computing, and helped define the modern studies of complex systems and workflow management.

3.1.1 Elements of Petri nets

A Petri net contains four types of elements: tokens, places, arcs, and transitions.

Tokens represent objects in the Petri net models. In a system modeling fish farming, typically a token represents a fish. A token is represented with a dot in Petri net models. When the number of tokens becomes large, it is usually represented with the number of tokens.

Places can hold tokens. Figure 2 shows places p_1, p_2, and p_3 with 4, 3m and 1 tokens (black spots). Each place is capable of holding any number of tokens.

Arcs are connections between places and transitions. Arcs are bipartite meaning it is not possible to have an arc connecting two places together or two transitions

Fig. 2 Sample Petri net

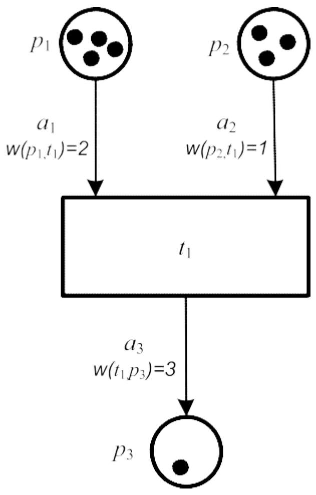
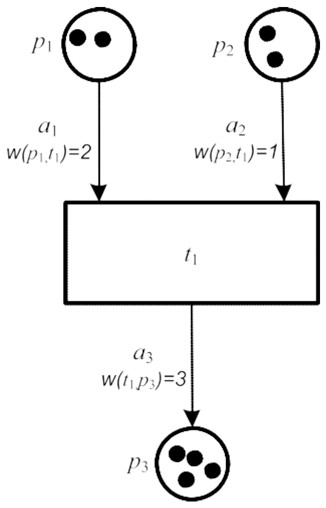


Fig. 3 Sample Petri net after one cycle



together. Each arc has a weight, which is the number of tokens that are transported simultaneously when the transitions of which the arc is connected to fires.

Transitions correspond to events and are connected by arcs to places. When a transition fires, the number of tokens within the places connected to the firing transition, are changed according to the arcs weights and directions; when a transition fires, it consumes tokens (input parts) from the input places and puts tokens (output parts) into the output places. For a transition to be able to fire, the number of tokens in the input places must be equal or higher than the weights of the arcs connecting the input places to the transition. The transition will then be an *enabled transition*. Figure 3 shows the state of the sample Petri net from Fig. 2 after the transition T1 has fired once.

3.1.2 *Formal definition of Petri nets*

A Petri net is a four-tuple (P, T, A, x_0) where

P is the set of places, $P = [p_1, p_2, \ldots, p_n]$
T is the set of transitions, $T = [t_1, t_2, \ldots, t_m]$
$A \subseteq (P \times T) \cup (T \times P)$ is a set of arcs from places to transitions and from transitions to places, and
$x = [x(p_1), x(p_2), \ldots, x(p_{n1})] \in N^n$ is the row vector of markings (tokens) on the set of places; x_0 is the initial marking.

3.1.3 *Input and output places of a transition*

In the Petri net in Fig. 3, the places p_1 and p_2 are inputs to transition t_1, and P3 is an out place of transition t_1. It is convenient to use $I(t_j)$ to represent the set of input places to transition t_j and $O(t_j)$ to represent the set of output places to transition t_j

when describing a Petri net:

$$I(T_j) = \{p_i \in P : (p_i, t_j) \in A\}$$
$$O(t_j) = \{p_i \in P : (p_i, t_j) \in A\}$$

We see from Fig. 3, that the weight of the arc from input place p_1 to transition t_1 has a weight = 2. This is denoted by: $w(p_1, t_1) = 2$.

3.1.4 Enabled transition

A transition $t_j \in T$ in a Petri net is said to be *enabled* if [12]:

$$x(p_i) \geq w(p_i, t_j) \quad \text{for all } p_i \in I(t_j).$$

The transition t_1 in Fig. 3 is enabled, since the numbers of tokens in the input places p_1 and $p_2(2)$ are at least as large as the weight of the arcs connecting them to $t_1(w(p_1, t_1) = 2$ and $w(p_1, t_1) = 2)$.

3.1.5 Petri net dynamics

The markings of a Petri net, which is the distribution of tokens to the places, represent the state of the Petri net. A Petri net representing a discrete event system, where the transitions represent events, goes through many states during a simulation process. The different states could be represented with the row vector of markings (the 4.th-tuple): $x = [x(p_1), x(p_2), \ldots, x(p_{n1})]$.

The number of states an *infinite capacity net* can have is generally infinite, since each place can hold an arbitrary nonnegative integer number of tokens [9]. A *finite capacity net* on the other hand, will have a given number of possible states.

The *state transition function*, $f : \aleph^n \times T \rightarrow \aleph^n$, of a Petri net is defined for a transition $t_j \in T$ if and only if, $x(p_i) \geq w(p_i, t_j)$ for all $p_i \in I(t_j)$.

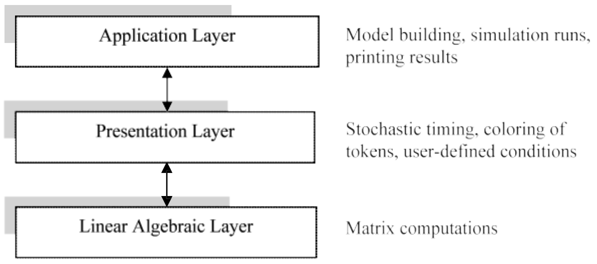
If $f(x, t_j)$ is defined then $x' = f(x, t_j)$, where

$$x'(p_i) = x(p_i) - w(p_i, t_j) + w(t_j, p_i), \quad i = 1, \ldots, n.$$

3.1.6 Why Petri nets

Several tools could be used for simulation of discrete event systems; Automata, Stateflow, and Petri nets (high level) are some of the most commonly used [5]. The lack of structure possibilities (hierarchy) in Automata is a serious shortcoming, since the Atlantic salmon farming industry is a complex system which should be a be decomposed into modules and sub systems [Harhalakis, G., Proth]. Stateflow, developed by The MathWorks, extends the Simulink part of MATLAB with functionality similar to Petri net; charts are used for graphical representation of hierarchical and parallel states and for the event-driven transitions between them [5]. A Petri net model of a discrete event system could easily be converted into a Stateflow model and vice versa, but learning Stateflow is much more difficult than learning Petri net due to the syntactic, semantic, and graphical details in Stateflow. Stateflow also requires some

Fig. 4 3-layer architecture


Application Layer
Model building, simulation runs, printing results
Presentation Layer
Stochastic timing, coloring of tokens, user-defined conditions
Linear Algebraic Layer
Matrix computations

knowledge of Simulink, in addition to MATLAB, while the GPenSIM tool used for Petri net simulation in this paper runs under the MATLAB environment only. Petri nets is widely accepted by the research community for modeling and simulation of discrete event-driven systems, mainly due to graphical representation and the well defined semantics which makes it possible to use formal analysis of the models [5].

3.2 GPenSIM

GPenSIM (General Purpose Petri net Simulator) is written in MATLAB language which allows seamless integration with the other toolboxes that also available in the MATLAB environment [11].

3.2.1 Architecture of GPenSIM

GPenSIM is designed using the well-proven paradigms in software engineering such as: layered architecture, modular components, and natural language interface.

GPenSIM is built following 3-layer architecture; see Fig. 4. The bottom layer deals with Petri net run-time dynamics; this layer computes newer states with the help of linear algebraic equations and matrix manipulations. The middle layer adds more high-level functionality such as stochastic timing, coloring of tokens, user-defined conditions ('guard-conditions' in some literature), etc. The top layer offers applications such building a Petri net based model, running simulations, determining coverability tree, printing the simulation results, etc.

3.2.2 Modular components

A model of a discrete event system developed with GPenSIM consists of a number of files. The main simulation file (MSF) is the file that will be run directly by the MATLAB platform. In addition to the main simulation file, there will be one or more Petri net definition files (PDFs); definition of a Petri net graph (static details) is given in the Petri net Definition File. There may be a number of PDFs, if the Petri net model is divided into many modules, and each module is defined in a separate PDF. While the Petri net definition file has the static details, the main simulation file contains the dynamic information (such as initial tokens in places, firing times of transitions) of the Petri net. In addition to these files (main simulation file and Petri net definition files), there can be a number of transition definition files (TDFs), also.

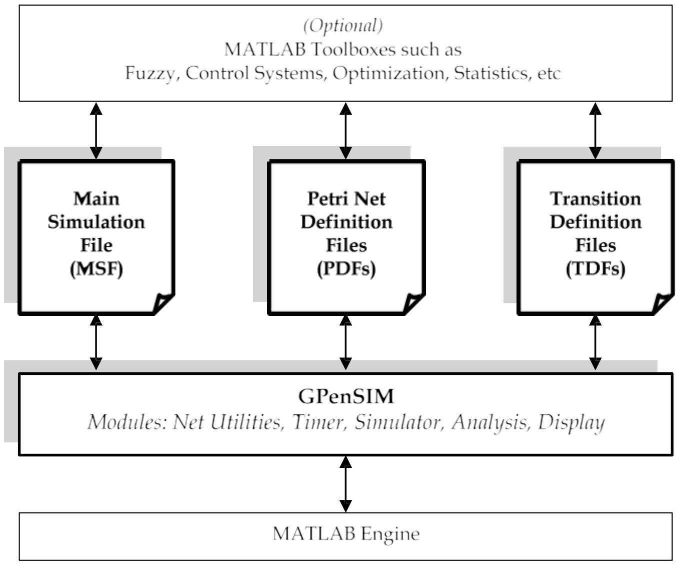

(Optional)
MATLAB Toolboxes such as
Fuzzy, Control Systems, Optimization, Statistics, etc
Main Simulation File (MSF)
Petri Net Definition Files (PDFs)
Transition Definition Files (TDFs)
GPenSIM
Modules: Net Utilities, Timer, Simulator, Analysis, Display
MATLAB Engine

Fig. 5 The architecture of GPenSIM

There are two types of transition definition file (TDF): preprocessor and post-processor. A preprocessor generally consists of additional conditions that determine whether an enabled transition can fire or not. The additional conditions are called "user defined condition" in GPenSIM terminology, whereas in some other literature (e.g., Colored Petri Net) it is referred to as "guard-functions"). There can be a separate preprocessor for each transition in a Petri net model. A post-processor generally do the tidying-up work or accounting work after firing of a transition.

3.2.3 Natural language interface

Users need not know Petri net mathematics when creating a Petri net model of a discrete event system. GPenSIM offers a natural language interface with which model building mainly deals with identifying the basic elements of a system and establishing the connections between these elements. Figure 5 shows the overall architecture of GPenSIM.

Figure 6 shows the main loop of the simulator. As in any Petri net simulator, the main loop consists of a simple cycle that first checks whether any transitions are enabled and then it puts the enabled transitions into the firing queue, provided that the transitions satisfy additional user defined conditions, if any; inputs tokens are also taken away (consumed) by the corresponding transitions. Then the loop checks whether any firing transitions are completing or have completed. In this case, the firing transitions are popped out of the firing queue and output tokens are deposited into the respective output places.

Figure 6 also shows that there are two kinds of timers are in use. The first timer—called global timer is the one that is normally used. The second timer called stochastic

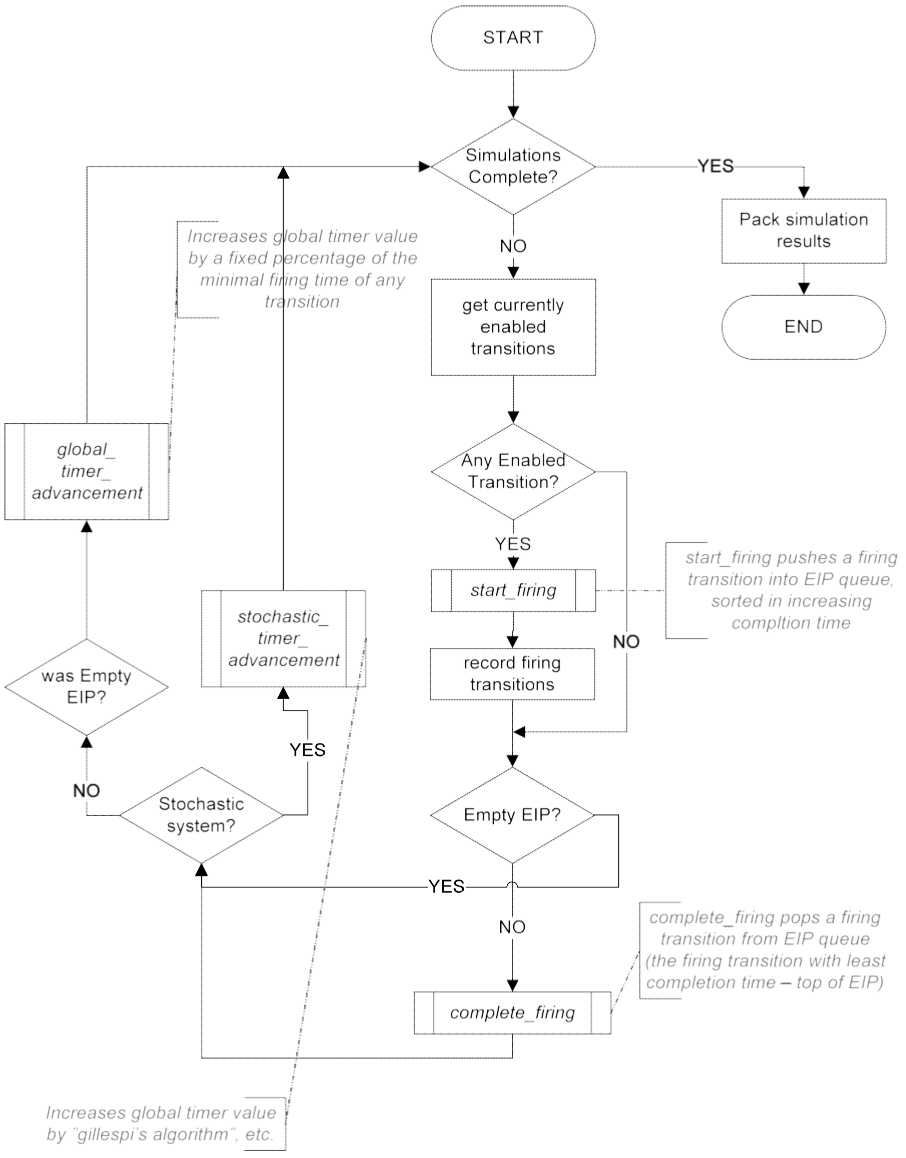

START
Simulations Complete?
YES
Pack simulation results
END
NO
get currently enabled transitions
Any Enabled Transition?
YES
start_firing
start_firing pushes a firing transition into EIP queue, sorted in increasing compltion time
record firing transitions
NO
Empty EIP?
YES
NO
complete_firing
complete_firing pops a firing transition from EIP queue (the firing transition with least completion time – top of EIP)
Stochastic system?
YES
NO
stochastic_ timer_ advancement
was Empty EIP?
global_ timer_ advancement
Increases global timer value by a fixed percentage of the minimal firing time of any transition
Increases global timer value by "gillespi's algorithm", etc.

Fig. 6 The main loop of the simulation runs

timer is used only for "stochastic systems." Stochastic systems can be leisurely defined as continuous systems (as opposed discrete systems) that are to be discretized first into discrete systems so that a Petri net model can be created for them. A case study on stochastic systems is performed in Sect. 5.

Finally, the main loop shown in Fig. 6 hints an extension to GPenSIM: by using a Real-Timer (computer's real-time clock) instead of stochastic or global timer, a Real-

Time GPenSIM version can be developed. This Real-Time GPenSIM is basically a soft Programmable Logic Controller (PLC), which will use a Digital & Analogue Input Output Card (DAC) to read sensor inputs from the outside world and will also output digital signals to triggers via the card. In this real.-time version, the main loop should read the sensor data at the start of each cycle, and the state of the firing transitions should be mapped to the output triggers.

3.2.4 Methodology for modeling and simulation with GPenSIM

Creating a Petri net model consists of two steps:

- Defining the static Petri net graph, and
- Assigning initial dynamics in the main simulation file.

Defining the Petri net graph in one or more Petri net Definition Files (PDF): this is the static part. This step consist of three sub-steps:

- Identifying the basic elements of a Petri net graph: the places,
- Identifying the basic elements of a Petri net graph: the transitions, and
- Connecting the elements with arcs.

Assigning the dynamics of a Petri net in the Main Simulation File (MSF):

- The initial markings on the places, and possibly
- The firing times of the transitions.

After creating a Petri net model, simulations can be done.

4 Using Petri nets

This section only deals with the stage-4 “global Optimization” of the approach given in the previous section. Due to space limitation, stages 1–3 are not discussed any further.

This paper uses Petri nets for analyzing workflows. Petri net is a widely accepted for modeling and simulation of discrete event-based systems.

A Petri net model of collaborative workflow can be huge and difficult to manage as the virtual enterprise may contain many layers of suppliers and distributors. Because of the huge size of the Petri net model, the model has to be built module-by-module using the modular approach.

The modular approach for Petri net model building consists mainly of modeling individual enterprises as modules, and then connecting the modules together through their input/output ports. The modular modeling approach is a three step process, the step are explained in the following subsections.

4.1 Step-1: collaborative enterprises as event graphs

In this first step, the collaborating enterprises are modeled as *event graph* modules, as event graphs can be easily reduced into minimal size using existing theories; model

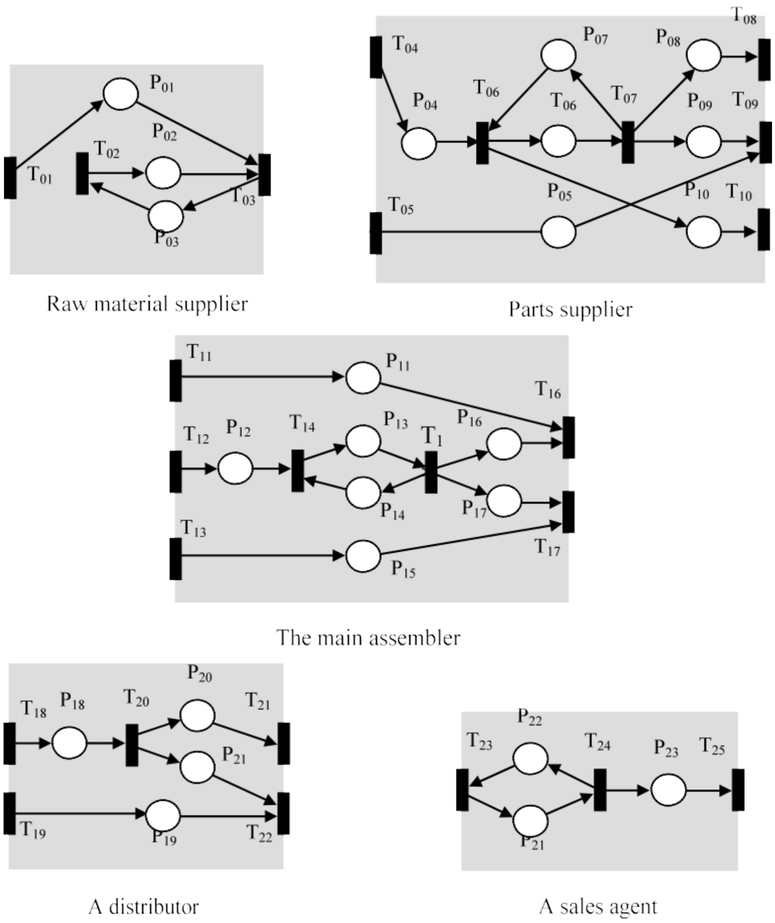


Fig. 7 Event graph module of different types of collaborating enterprises

reduction will be done in step-2. The event graph modules must also posses Input/Output ports (I/O ports); it is only through the I/O ports, the modules are going to be connected with the other modules in order to obtain the complete model.

Figure 7 shows the modules of different types of collaborating enterprises as event graphs; event graphs are Petri nets models in which all the places have exactly one input transition and one output transition. The description of places and transitions of the modules shown in Fig. 7 is given in Table 1.

4.2 Step-2: reducing event graph modules

The second step is to reduce the size of the modules; Savi and Xie [16] and Harhalakis et al. [12] show a reduction method that is suitable for reducing event graph modules surrounded by I/O ports (transitions). According to the reduction theorem, the minimal representation (reduced model) has the same set of input and output ports

Table 1 Description of the places and transitions

Raw material supplier	
T_{01}	Receive order from part supplier/assembler
T_{02}	Production of raw material
T_{03}	Ship material to part manufacturer
P_{01}	Orders for material received
P_{02}	Material ready for shipment
P_{03}	Volume of material to produce
Part supplier	
T_{04}	Material arrive (from mat. supplier)
T_{05}	Orders received from assembler
T_{06}	Production calculations
T_{07}	Manufacture of parts
T_{08}	Order raw materials
T_{09}	Ship parts to main assembler
T_{10}	Send bid to main assembler
P_{04}	Unloaded material in queue
P_{05}	Received orders for parts
P_{06}	Ready for production of parts
P_{07}	Monitor of quantity produced
P_{08}	Monitor of materials used
P_{09}	Parts ready for shipment
P_{10}	Bid for parts (to assembler)
Main assembler	
T_{11}	Bids for parts and raw materials
T_{12}	Parts and raw materials arrive
T_{13}	Orders received
T_{14}	Production calculations
T_{15}	Manufacture of products
T_{16}	Order parts and raw materials
T_{17}	Ship products to distributors
P_{11}	Received bids from suppliers
P_{12}	Unloaded parts in queue
P_{13}	Ready for manufacture
P_{14}	Monitor of quantity produced
P_{15}	Received orders for products
P_{16}	Monitor of parts/materials used
P_{17}	Products ready for shipment
Distributor	
T_{18}	Goods arrive from main assembler
T_{19}	Receive orders for goods from sales agent
T_{20}	Supply and distribution calculation
T_{21}	Order goods from main assembler
T_{22}	Ship goods to sales agents
P_{18}	Stock of goods
P_{19}	Received orders for goods
P_{20}	Monitor of free capacity
P_{21}	Goods ready for shipment
Sales agent	
T_{23}	Goods arrive from distributor
T_{24}	Sales
T_{25}	Order goods from distributor
P_{21}	Stock of goods
P_{22}	Monitor of free capacity
P_{23}	Monitor of sales

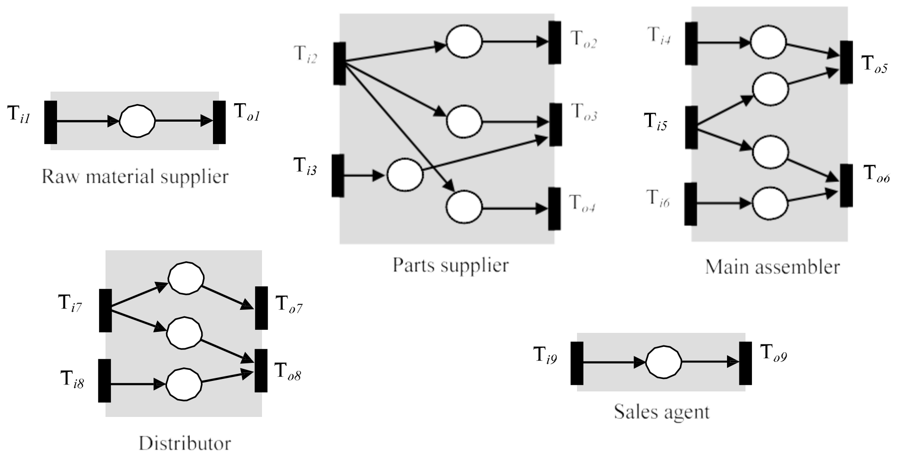


Fig. 8 Minimal representations for different types of enterprises

as the initial module, but has fewer internal places and do not has any internal transitions [16]. Therefore, the removal of internal transitions and reduction in internal places greatly reduces the overall size of the complete model.

By the reduction theorem, though simple event graph module called minimal representations replaces more complex modules of different kinds of enterprises, the liveness and boundedness properties of the original modules are preserved.

Figure 8 shows the minimal representations of the event graph modules that are shown in Fig. 7.

4.3 Step-3: connecting event graph modules together

The third step is to connect the modules together to form a complete model of the virtual enterprise. The modules are connected through their I/O ports. Since the I/O ports are transitions, the connecting line should inject a place in between the modules.

4.4 Working example

Figure 9 shows a collaborative workflow of a virtual enterprise consisting of 12 collaborating enterprises. Since the working example is given as a proof-of-concept, it is intentionally made simpler with just 12 collaborating enterprises. *Not* shown in Fig. 9: the event graph modules for the transporting agents are unique as these modules have only one I/O port as this port functions as both input and output port. Technically, this I/O port is a single transition. Also, the market is represented by an event graph module consisting of a single input port; there is no output port in this module, and hence it functions as “sink”—consuming whatever passed to it.

The tool *General Purpose Petri net Simulator* (GPenSIM) [11] is used for simulation. GPenSIM uses colors to detect possible redundancy in the virtual enterprising. For more details on coloring in Petri nets, interested readers are referred to the GPen-SIM user manual [11].

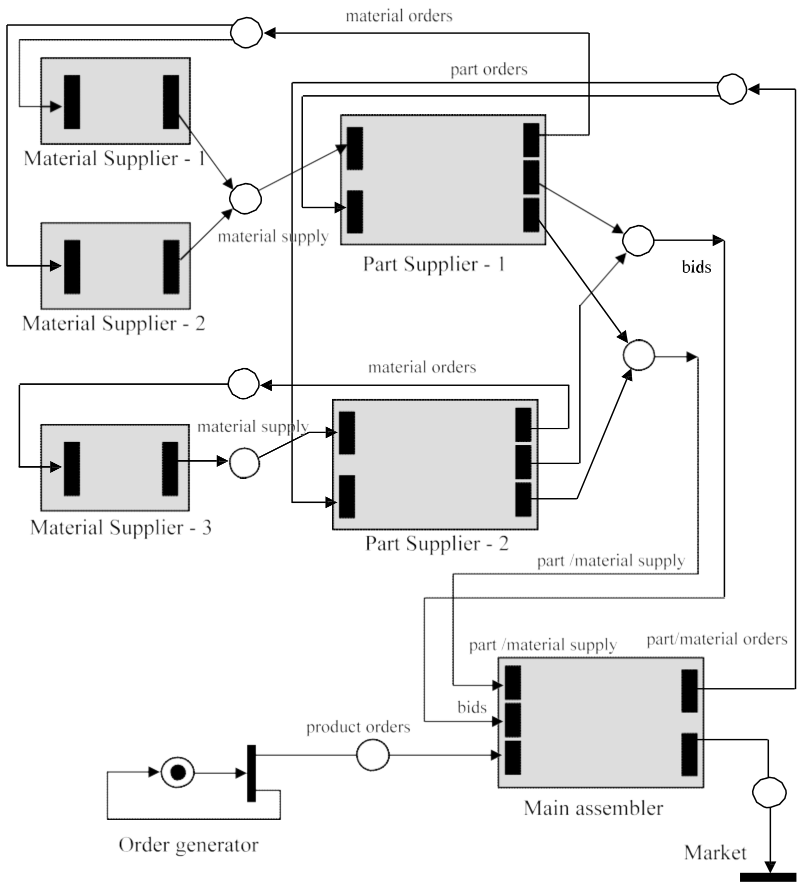


Fig. 9 Model of a virtual enterprise consisting of material suppliers, part suppliers and the main assembler

4.4.1 Simulation program

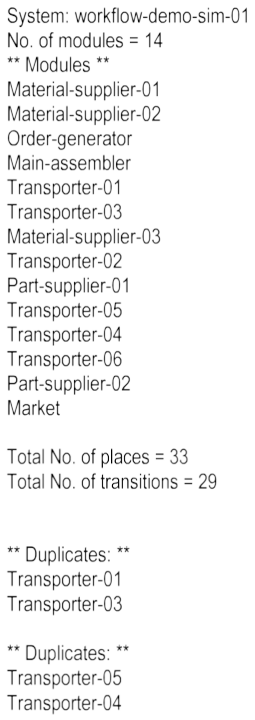
The simulation program consists of 14 modules (12 modules, one for each collaborating enterprises; one module for the market, and one for the main simulation file that also posses the order generators and the buffer places between the modules. Due to space restriction, we present the 2-line code snippet for simulation run:

```
>> [simulation_results, colormap] = gpensim(workflow_demo, dynamic_info);
>> printsys(simulation_results, 'summary');
>> print_colormap(workflow_demo, colormap, 'duplicates');
```

4.4.2 Simulation results

Simulation results, shown in Fig. 10, depict that there could be two sets of redundant functions:

Fig. 10 Simulation results



```
System: workflow-demo-sim-01
No. of modules = 14
** Modules **
Material-supplier-01
Material-supplier-02
Order-generator
Main-assembler
Transporter-01
Transporter-03
Material-supplier-03
Transporter-02
Part-supplier-01
Transporter-05
Transporter-04
Transporter-06
Part-supplier-02
Market

Total No. of places = 33
Total No. of transitions = 29


** Duplicates: **
Transporter-01
Transporter-03

** Duplicates: **
Transporter-05
Transporter-04
```

- Transporting agent 01, and Transporting agent 03 are perhaps transporting materials in the same region, thus can be combined. Similarly,
- Transporting agent 05, and Transporting agent 04 are perhaps transporting materials in the same region, thus can be combined.

5 Distributed workflow simulation

Though the approach presented above is for a distributed system like virtual enterprising where individual enterprises work independently, the simulation program assumes that all the modules for simulation are available in one place. In order to perform a truly distributed simulation, where individual modules are modeled, simulated, and optimized individually, before they are combined, and ready for iterative optimization cycle, the tool for modeling and simulation must provide distributed modeling and simulation capabilities.

5.1 Extending GPenSIM to provide distributed capabilities

Though GPenSIM originally was designed with distributed applications in mind, the current version is not suitable for distributed applications, yet. This is because the support in MATLAB for distributed applications was not satisfactory until recently. MATLAB did not support multithreading: it did not support starting parallel sessions

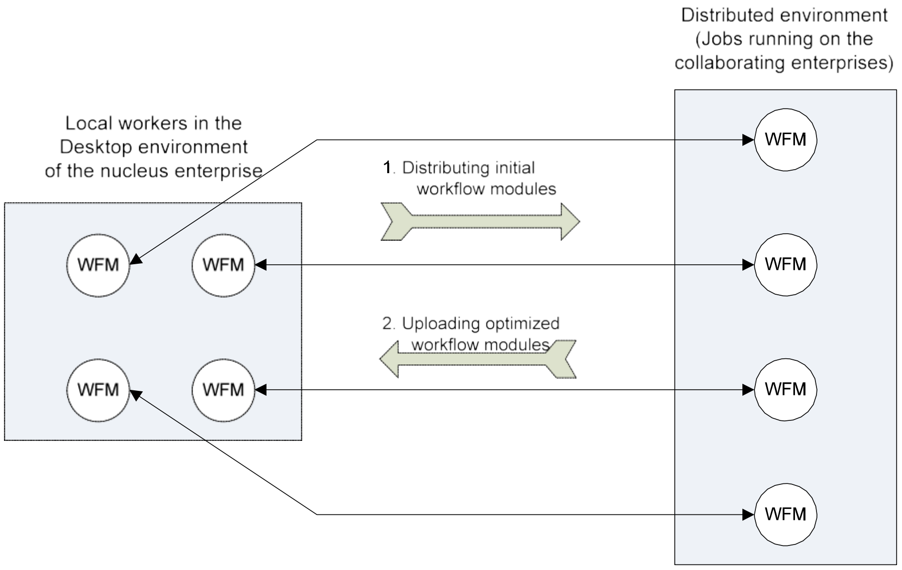

Distributed environment (Jobs running on the collaborating enterprises)
Local workers in the Desktop environment of the nucleus enterprise
1. Distributing initial workflow modules
2. Uploading optimized workflow modules
WFM

Fig. 11 Distributed approach for optimizing workflow modules

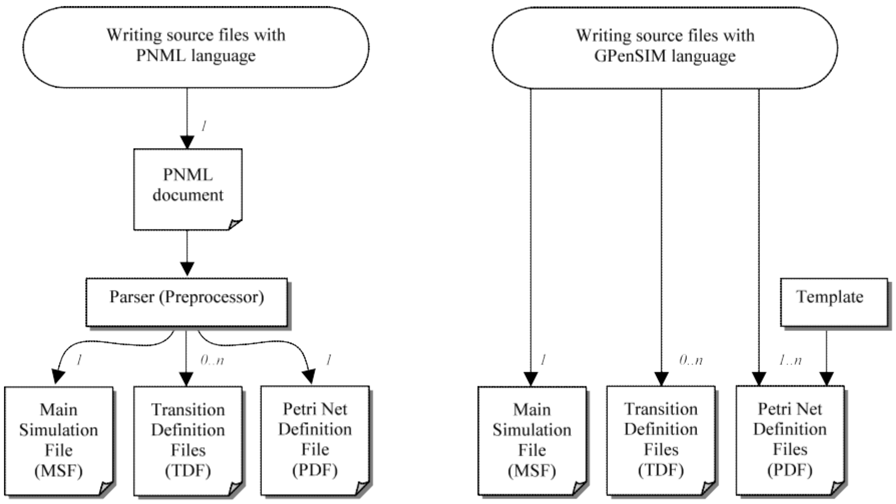
within the same program, so that these parallel sessions can really run in parallel and function as proxies for distributed applications running elsewhere. However, MATLAB has now provided a new toolbox called Parallel Computing Toolbox [14]. With Parallel Computing Toolbox, implementing a distributed GPenSIM has become a distinct possibility. To make a distributed GPenSIM version and to use it for simulating redundancy elimination in virtual enterprising, there are some improvements needed to the GPenSIM architecture.

5.2 Using local workers as proxies for distributed applications

Figure 11 shows distributed approach:

- Stage-1: Once the crude virtual-enterprise wide global workflow is made, it is portioned into workflow modules for the individual enterprises; these modules are loaded into the local workers that are running on the desktop system of the nucleus enterprise.
- Stage-2: Partitioned workflow modules inside the local workers and sent to distributed applications that are run on the platforms of collaborating enterprises.
- Stage-3: After optimizing workflow modules individually, collaborating enterprises sent their optimized workflow modules back to the nucleus enterprise.
- Stage-4: Global optimization: The individual locally optimized workflow modules are now available in the local workers are joined together to form the collaborative workflow.

There are some restrictions on the number of local workers that can run in parallel as the current version of the MATLAB Parallel Computing Toolbox set the maximum



Writing source files with PNML language
1
PNML document
Parser (Preprocessor)
1
0..n
1
Main Simulation File (MSF)
Transition Definition Files (TDF)
Petri Net Definition File (PDF)
Writing source files with GPenSIM language
Template
1
0..n
1..n
Main Simulation File (MSF)
Transition Definition Files (TDF)
Petri Net Definition Files (PDF)

Fig. 12 Two different approaches for coding distributed workflow modules

number of local workers that can run in parallel to four. This is indeed a serious problem, as a virtual enterprises could constitute many collaborating enterprises.

5.3 PNML as the format for exchanging Petri net modules

Virtual enterprises' heterogeneous platform—a huge diversity of programming languages and operating systems—is a barrier to distributed workflow management. Thus, the communication between platforms must be based on a language and operating system neutral standard, meaning the modeling and simulation tool GPenSIM should allow workflow modules and messaging between the modules in a standard interchange format, such as XML based PNML (Petri Net Markup Language) [15].

5.3.1 Using PNML for workflow modules

Figure 12 shows two approaches for coding workflow modules. 1) Using PNML (shown on the left-hand side of Fig. 12), and 2) Using GPenSIM language (shown on the right-hand side of Fig. 12).

By the first approach, the *overall* Petri net model of a system is described in a single source file is called is PNML document, using PNML language; PNML document is then fed to GPenSIM parser, which will convert the PNML document into a set of source files (one MSF, one PDF, and zero or more TDF). The current version of GPenSIM does not allow the use of PNML to model subsystems as separate Petri nets, meaning there will be only one PDF generated by the parser. By the second approach, all the source files (MSF, PDFs, TDFs) are coded in GPenSIM language.

5.4 Summary

This section presents the programming language and techniques for extending GPenSIM to support distributed workflow simulations. This work is not complete yet. The main performance problem that is visible at this stage is the scalability: MATLAB Parallel Computing Toolbox allows a maximum of 3 parallel processes per node. This means, for simulation on *n* virtual enterprises, at least *n/*3 computing nodes must be utilized.

6 Conclusion

This paper shows a new approach for detection and elimination of redundancy in virtual enterprises. The approach uses distributed workflows. The approach uses an open tool known as GPenSIM that is available for the industry standard modeling and simulation platform MATLAB; in addition this approach proposes the use of platform and language neutral PNML language for coding the workflow modules.

Literature review reveals that distributed workload management is not a new concept [4, 9, 13]. However, all the papers available on this topic use their own propriety systems and not an open industry standard platform like the one proposed in this paper. This paper emphasizes use of open standard tools as the approach requires co-ordination across diverse IT groups, including strategy, development, and datacenter operations. In many enterprises, these organizations rely on different tools, platforms, and policies and have limited points of shared decision making and policy development. As a result, there is some risk that distributed workload solutions will struggle to find internal optimized sub-solutions, unless the workloads and applications are fully packaged for portability.

References

1. Badir YF, Buchel B, Tucci ChL (2005) The role of the network lead company in integrating the NPD process across strategic partners. Int J Entrepreneurship Innov Manag 5(1–2):117–137
2. Brandic I, Pllana S, Benkner S (2008) Specification, planning, and execution of QoS-aware grid workflows within the Amadeus environment. Concurr Comput Pract Experience 20(4):331–345
3. Cassandaras CG, Lafortune SL (1999) Introduction to discrete event systems. Kluwer Academic, Norwell, ISBN 0-7923-8609-4
4. Churches D, Gombas G, Harrison A (2005) Programming scientific and distributed workflow with Triana services. Concurr Comput Pract Experience 18(10):1021–1037
5. Davidrajuh R (2000a) Automating supplier selection procedures. PhD Thesis, Norwegian University of Science & Technology, Trondheim, Norway; ISBN: 82-7984-159-8, ISSN: 1081-1393
6. Davidrajuh R (2000b) A Petri net approach for performance measurement of supply chain in agile virtual enterprise. In: MIS Rev, vol 10, December 2000, ISSN: 1081-1393
7. Foster I, Kesselman C, Nick JM, Tuecke S (2002) Grid services for distributed system integration. Computer 35(6):37–46
8. Foster I, Kesselman C, Tuecke S (2001) Int J High Perform Comput Appl 15(3):200–222
9. Geppert A, Tombros D (2009) Event-based distributed workflow execution with EVE. In: Proceedings of the IFIP international conference on distributed systems platforms and open distributed processing, The Lake District, United Kingdom, pp 427–442
10. Gil Y, Deelman E, Ellisman M, Fahringer T, Fox G, Gannon D, Goble C, Livny M, Moreau L, Myers J (2007) Computer 40(12):24–32
11. GPenSIM (2010) http://www.davidrajuh.net/gpensim
12. Harhalakis G, Proth JM, Savi VM, Xie X (1991) A stepwise specification of a manufacturing system using Petri nets. In: Proceedings of the 1991 IEEE international conference on systems, man and cybernetics, Charlottesville, Virginia, October 1991

13. Kochut K, Arnold J, Sheth A, Miller J, Kramer E, Arpinar B, Cardoso J (2003) IntelliGEN: a distributed workflow system for discovering protein-protein interactions. J Distrib Parallel Databases 13(1):43–72
14. MATLAB (2010) Parallel Computing Toolbox; Available: http://www.mathworks.com
15. PNML (2010) Available: http://www2.informatik.hu-berlin.de/top/pnml/
16. Savi VM, Xie X (1992) Liveness and boundedness analysis for Petri nets with event graph modules. In: Petri nets. Lecture notes in computer science series. Springer, Berlin
17. Van der Aalst W, Weske M (2001) The P2P approach to interorganizational workflows. In: Lecture notes in computer science. Springer, Berlin
18. Yu J, Buyya R (2005) A taxonomy of workflow management systems for grid computing. J Grid Comput 3(3):171–200
19. Zur Muehlen M (2004) Organizational management in workflow applications—issues and perspectives. Inf Technol Manag 5(3):271–291