Scheduling of Sequence-Dependant Jobs on Parallel Multiprocessor Systems Using a Branch and Bound-Based Petri Net

Abolfazl Jalilvand
Department of Electrical Engineering
Zanjan University
Zanjan, Iran
ajalilvand@tabrizu.ac.ir

Sohrab Khanmohammadi
Faculty of Electrical and Computer Engineering
University of Tabriz
Tabriz, Iran
khan@tabrizu.ac.ir

Fereidoon Shabaninia
Faculty of Electrical and Electronics Engineering
University of Shiraz
Shiraz, Iran
shabani@shirazu.ac.ir

Abstract

This paper presents a new branch-and-bound algorithm for minimizing the make-span of a job-shop scheduling problem where n jobs must be scheduled on m machines. It is assumed that the jobs are available at time zero and have sequence-dependent setup times on machines. For solving the scheduling problem we develop a new Branch and Bound system which constructs its search tree gradually and does not need a large size memory. An upper-bound cost (UBC) is introduced to initialize the root node in search tree which reduces Branch and Bound computations .For modeling the manufacturing system and applying the desired sequence-dependent schedule on it, a supervisor Petri net is introduced. The proposed methods will be verified through a computational experiment.

Keywords: Multiprocessor system, Scheduling, Petri nets, Branch and Bound.

1. Introduction

Applications of sequence-dependent scheduling are commonly found in manufacturing environments. In the printing industry, for example, presses must be cleaned and settings changed when ink color, paper size or receiving medium differ from one job to the next. In the container manufacturing industry machines must be adjusted whenever the dimensions of the containers are changed, while in printed circuit board assembly, rearranging and restocking component inventories on the magazine rack is required between batches. In each of these situations, sequence-dependent setup times play a major role and must be considered explicitly when modeling the problem [1].

The search for a solution to the multiprocessor scheduling problem is performed with the aid of a *search tree* that represents the solution space of the problem. That is, all possible permutations of task–to–processor assignments and schedule orderings. The Branch-and-Bound (B&B) strategy has been successfully used for searching such a solution space and finding optimal or near-optimal solutions [2-4].

In a manufacturing system, for example, many machines may be able to process a job but with different speeds depending on the machine's technology. For such a system when no precedence constraints exist between tasks, the number of goal vertices in a search tree is $n!m^n$ for a multiprocessor system with n tasks and m processors [2]. In such a system constructing the search tree thoroughly needs $\sum_{k=1}^{n} m^k \prod_{l=0}^{k-1}(n-l)$ memory cells, so that even for low number of jobs and machines, using B&B algorithm needs a large size of memory.

In this paper we introduce a new method which enables us to construct the search tree gradually. Using this new algorithm vertices are generated just when the B&B algorithm needs to explore them. For initializing the root node in the search tree a heuristic upper bound cost will be introduced which reduces the branch-and-bound computations. To apply the desired schedule on the manufacturing system a supervisor Petri net is used.

2. The Multiprocessor System

The multiprocessor system consists of a set of processors (manufacturing cells) indexed by $i \in I = \{1,2,...,m\}$ and a set of jobs, indexed by $j \in J = \{1,2,...,n\}$ where each of jobs can be processed by each of the machines. It is assumed that the required processing time of each job is different for each of machines. Furthermore it is assumed that there are sequence-dependent setup times on machines. The main problem is to determine which sequence of jobs must be processed by any machine so that overall make-span of the system is minimized. We can summarize all of the (processing + setup) times in m $n \times n$ time matrices as follows:

$$T^i = [t^i_{kj}]_{n \times n} : i \in I = \{1 \quad 2 \quad ... \quad m\} \qquad (1)$$

Where t^i_{kj} represents the (setup + processing) time of i^{th} machine on j^{th} job whereas it has processed k^{th} job already. If a machine isn't eligible to process a specific job it can be considered by setting the elements of the corresponding row and column in it's time matrix to infinity. We introduce C_i as the completion time of the last job scheduled on i^{th} machine which can be defined as follows:

$$C_i = \sum_{j=1}^{N_i} t^i_{kj} \qquad (2)$$

Where N_i is number of jobs scheduled on machine i and t^i_{kj} is the (setup + processing) time of job j on machine i whereas it has processed k^{th} job already. By using all C_i the make-span ($C_{\max}$) is defined as:

$$C_{\max} = \max_{i \in I}(C_i) \qquad (3)$$

Where the objective of scheduling is to minimize $C_{\max}$.

3. Branch and Bound Algorithm

Branch and Bound is a common search technique for combinatorial optimization. B&B improves over exhaustive enumeration, because it avoids the exploration of those regions of the solution space, where it can be certified (by means of lower bounds) that no solution improvement can be found. The exploration of the solution space can be represented by a search tree where its nodes represent sets of solutions, which can be further partitioned in mutually exclusive sets. Each subset in the partition is represented by a child of the original node. Whenever a new vertex is generated which could lead to an optimal solution, it will be referred to as an *active vertex*. The power of the B&B strategy lies in alternating branching and bounding operations on the set of active vertices. The *branch* refers to partitioning of the solution space (generating the child vertices); the *bound* refers to lower bounds that are used to construct a proof of optimality without exhaustive search (process of evaluating the cost of new child vertices). A *goal vertex* in the search tree represents a complete solution where all tasks have been scheduled on the processors. An acceptable complete solution is also called a *feasible* solution. An *intermediate vertex* represents a partially complete schedule. The *level* of a vertex is the number of tasks that have been assigned to any processor in the current schedule. The *cost* of a vertex is the quality of the schedule represented by the vertex (Fig. 1).

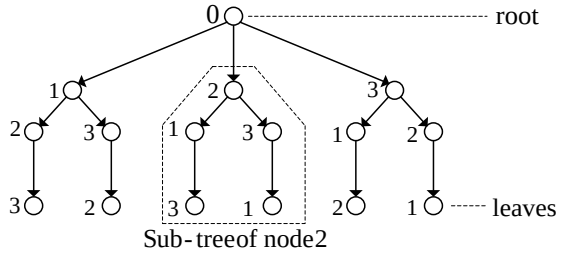

Fig.1: The structure of a tree.

The B&B algorithm starts by sequencing one of the available operations called branching the node. By branching a node, a new node is formed and the node is kept in the search space if its lower bound value is better than the upper bound value or vice versa. A heuristic is used to schedule the remaining operations for every node and the best solution found so far will be recorded as the upper bound value. The next node to be branched is the one with the best lower bound value, as it is deemed to have the best potential. As more nodes are branched, more and more operations will be sequenced, and the upper bound value will become smaller and smaller [6]. An algorithm that computes a lower bound on the cost of any solution in a given subset prevents further searches from a given node if the best cost found so far is smaller than the cost of the best solution that can be obtained from the node (lower bound computed at the node). In this case the node is killed and no children need to be searched; otherwise it is alive.

It is well known that Branch-and-Bound computations can be reduced by using a heuristic to find a good solution to act as an upper bound prior to the application of the enumeration algorithm, as well as at certain nodes of the search tree [4]. The more accurate the upper-bound cost is, the faster the B&B algorithm will get because more vertices can be pruned at each step. For the mentioned scheduling problem we introduce a heuristic UBC as follows:

Consider the jobs one by one from 1 to n. Related to each job, after adding the make-spans of scheduled machines to the corresponding (setup + processing) times related the new job we select the machine with minimum overall time and consider it's index as the machine that the corresponding job is assigned to it. When this process is completed we have a sequencing of machines related to jobs. Now we calculated the make-span of each machine and based on them we compute $C_{\max}$ of this sequencing as a UBC using (3).

4. The New B&B Algorithm

In this section we introduce a new B&B algorithm based on a new method which constructs the search tree step by step. Constructing the search tree for this system starts with from the root (zero level). At the first level of the tree, mn branches are created. At level l of this tree, each node contains l jobs assigned, and can be branched into $m(n-l)$ nodes. When the last level of the tree is reached, the number of branches will be zero because at that point $l=n$. If this procedure is carried out completely, $n!m^n$ nodes will be generated at level n. For constructing whole of the search tree for this multiprocessor system we need $\sum_{k=1}^{n}(\prod_{l=0}^{k-1}(n-l))m^k$ memory cells. In our new approach, instead of constructing the search tree thoroughly, we just need two $n\times 1$ vectors ($\boldsymbol{D}$ and $\boldsymbol{B}$) and one $n\times n$ matrix $\boldsymbol{A}$ which reduce this high size of memory to only $n(n+2)$ memory cells. The key idea in our proposed B&B algorithm is to generate nodes of the search tree one by one and just when the B&B algorithm needs to explore it. At first we introduce an algorithm which can produce each of the possible combinations of m machines which n jobs assigned on them, one by one.

Algorithm I:

1- Get n as the number of jobs and m as the number of machines.
2- Set $\boldsymbol{D}=[1\ 1\ \dots\ 1]^T$ as the initial value of $\boldsymbol{D}$ and set $f=n$ as a temporary flag.
3- Set $\boldsymbol{D}(f)=\boldsymbol{D}(f)+1$.
4- IF $\boldsymbol{D}(f)>m$ and $f>1$ THEN set $\boldsymbol{D}(f)=1$ and $f=f-1$ then go to step 3 ELSE go to step 5.
5- End.

This algorithm only needs a $n\times 1$ vector $\boldsymbol{D}$ hence the size of required memory reduces considerably. Now we introduce another Algorithm for producing each permutation of the $n!$ distinct sequences at each step. Using this Algorithm each permutation of jobs is developed by just using a $n\times n$ matrix ($\boldsymbol{A}$) and a $n\times 1$ vector ($\boldsymbol{B}$) [9].

Algorithm II:

1- Select an arbitrary sequence of the n jobs as the first permutation and save it in array $\boldsymbol{S}$.
2- Set $\boldsymbol{B}=[1\ 2\ \dots\ n]^T$ as the initial value of $\boldsymbol{B}$, $\boldsymbol{A}=\left[\boldsymbol{S^T}\ \ \boldsymbol{S^T}\ \dots\ \boldsymbol{S^T}\right]^T$ as the initial value of $\boldsymbol{A}$ and $F=n$ as a temporary flag.
3- Set $\boldsymbol{P}=\boldsymbol{A}(n,:)$ where $\boldsymbol{P}$ indicates the permutation of $\boldsymbol{S}$ and $\boldsymbol{A}(n,:)$ indicates the n^{th} row of $\boldsymbol{A}$.
4- Set $\boldsymbol{B}(F)=\boldsymbol{B}(F)+1$.
5- IF $\boldsymbol{B}(F)>n$ and $F>1$ THEN set $\boldsymbol{B}(F)=F$, set $F=F-1$ and return to step 4 ELSE go to step 6.
6- Exchange $\boldsymbol{A}(F,\boldsymbol{B}(F))$ by $\boldsymbol{A}(F,F)$. Set rows F+1 to N of $\boldsymbol{A}$ equal to $\boldsymbol{A}(F,:)$.
7- IF $\boldsymbol{B}(F)\le n$ THEN set $F=n$, go to step 3 ELSE go to step 8.
8- End.

Now based on algorithms I and II we introduce a new B&B algorithm to solve the mentioned scheduling problem.

Algorithm III:

1- Get n as the number of jobs, m as the number of machines and $T^i: i\in I$ as the matrix that includes machining (setup + processing) times on jobs.
2- Select an arbitrary sequence of the n jobs as the first permutation and save it in array $\boldsymbol{S}$.
3- Set $\boldsymbol{B}=[1\ 2\ \dots\ n]^T$ as the initial value of $\boldsymbol{B}$, $\boldsymbol{A}=\left[\boldsymbol{S^T}\ \ \boldsymbol{S^T}\ \dots\ \boldsymbol{S^T}\right]^T$ as the initial value of $\boldsymbol{A}$, $F=n$ as a temporary flag.

4- Set $\boldsymbol{D}=[1\ 1\ \dots\ 1]^T$ as the initial value of $\boldsymbol{D}$ and set $f=n$ as a temporary flag.

5- Get $\boldsymbol{A}(n,:)$ as the sequence of jobs and $\boldsymbol{D}$ as the sequence of machines which are assigned jobs respectively and Calculate the make-span ($C_{\max}$) based on formula (3) then set MC (minimum make-span) equal to $C_{\max}$.

6- Get $\boldsymbol{A}(n,1:f)$ as the sequence of jobs and $\boldsymbol{D}(1:f)$ as the sequence of machines which are assigned jobs respectively and calculate the make-span ($C_{\max}$).

7- IF $C_{\max} \geq MC$ THEN go to step 9 ELSE go to step 8.

8- IF $f=n$ THEN set $MC=C_{\max}$ and go to 9 ELSE set $f=f+1$ and return to step 6.

9- Set $\boldsymbol{D}(f)=\boldsymbol{D}(f)+1$.

10- IF $\boldsymbol{D}(f)>m$ and $f>1$ THEN set $\boldsymbol{D}(f)=1$ and $f=f-1$ then go to step 9 ELSE go to step 11.

11- IF $\boldsymbol{D}(f)\leq m$ THEN go to step 6 ELSE go to step 12.

12- Set $\boldsymbol{B}(F)=\boldsymbol{B}(F)+1$.

13- IF $\boldsymbol{B}(F)>n$ and F>1 THEN set $\boldsymbol{B}(F)=F$ then set F=F-1 and return to step 12 ELSE go to step 14.

14- Exchange $\boldsymbol{A}(F,\boldsymbol{B}(F))$ by $\boldsymbol{A}(F,F)$. Set rows F+1 to N of $\boldsymbol{A}$ equal to $\boldsymbol{A}(F,:)$.

15- IF $\boldsymbol{B}(F)\leq n$ THEN set F=n and $\boldsymbol{D}=[1\ 1\ \dots\ 1]^T$, go to step 6 ELSE go to step 16.

16- End.

This B&B method performs a depth first search in an exhaustive manner. In this algorithm each node is constructed when it must be tested and there isn't need to construct the whole search tree at once.

For modeling of the manufacturing systems and to apply the optimum sequence which is obtained trough B&B algorithm we can use Petri nets [10].

5. Petri Nets

A Petri Net (PN) is a 5-tuple, $PN=(P,T,F,W,M_0)$ where [11]:

$P=\{p_1\ \ p_2\ \dots\ p_m\}$ is a finite set of places.

$T=\{t_1\ \ t_2\ \dots\ t_n\}$ is a finite set of transitions.

$F\subseteq(P\times T)\cup(T\times P)$ is a set of arcs (flow relations)

$W: F\rightarrow\{1\ \ 2\ \ 3\ \dots\}$ is a weight function.

$M_0: P\rightarrow\{0\ \ 1\ \ 2\ \dots\}$ is the initial marking.

$P\cap T=\Phi$ and $P\cup T\neq\Phi$.

The dynamical behavior of a system is modeled by changing the state or marking in Petri nets according to the following (firing) rules:

1- A transition t is said to be enabled if each input place p of t is marked with at least $w(p,t)$ tokens, where $w(p,t)$ is the weight of the arc from p to t.

2- An enabled transition may or may not fire depending on whether or not the event actually takes place (firing conditions are ok).

3- Firing of an enabled transition t removes $w(p,t)$ tokens from each input place p to t and adds $w(t,p)$ tokens to each output place p of t, where $w(p,t)$ and $w(t,p)$ are the weights of the arcs from p to t or t to p respectively.

In graphical representation of a Petri net, places are represented by circles and transitions are shown by hollow bars. The relationship between places and transitions is represented by directed arcs. For example the Petri net of Fig. 2 depicts the firing of a transition.

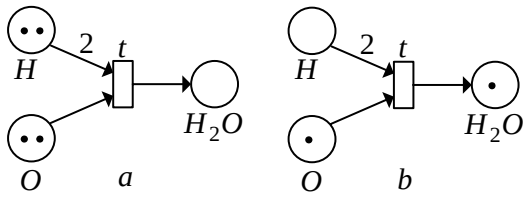

Fig.2: Transition (firing): (a) Marking before firing (b): Marking after firing.

In un-timed Petri net one can prohibit controlled transition from firing but cannot force the firing of a transition at a particular time. In a timed Petri net controlled transitions are forced to fire, by considering the time dependent firing functions. In timed Petri nets, each transition has its specific time which determines the transition's holding time. When a transition is fired during its holding time the network's marking is not changed and as soon as its holding time elapsed the marking of network will be changed based on the mentioned firing rules [12].

6. Applying B&B Using Petri Nets

For modeling the mentioned multiprocessor job-shop system it is considered that each

machine has an input buffer and an output buffer as shown in Fig. 3.

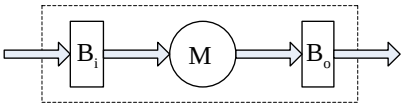

Fig. 3: Schematic diagram of a machine

Fig. 4 shows the Petri net model of each machine regarding its input and output buffers.

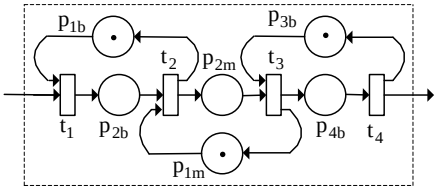

Fig.4: Petri net model of the machine.

In this model the places are defined as:
p_{1b}: the input buffer is empty.
p_{2b}: the input buffer is full.
p_{3b}: the output buffer is empty.
p_{4b}: the output buffer is full.
p_{1m}: the machine is idle.
p_{2m}: the machine is busy.
Also each transition in Fig. 4 is defined as:
t_1: A part enters the input buffer.
t_2: A part enters the machine.
t_3: The part exits the machine.
t_4: The part leaves the output buffer.

Based on this Petri net model we can model the multiprocessor system by considering *m* identical models which are parallel together. For applying the optimum sequencing obtained by B&B algorithm we introduce a Petri Net model. Fig. 5 shows this Petri net model. In this model related to each job we consider a place (p_j , $j = 1,2,...,n$). Regarding that each job can be assigned to each of the machines hence we consider *m* output arcs from each of these places (one output arc is related to one machine). To apply the optimum sequence it is needed to determine which job will be assigned to each machine. For this purpose we consider *m* additional places p_{ji} , $i = 1,2,...,m$ for each job j (totally $n \times m$ places). We can apply the desired sequencing by putting one token in related place. In this Petri net model we can assign the (setup + processing) time of each job by each machine to corresponding transition from the set of transitions labeled: t_{ji} , $j = 1,2,...,n$, $i = 1,2,...,m$. In such a case other transitions will be un-timed.

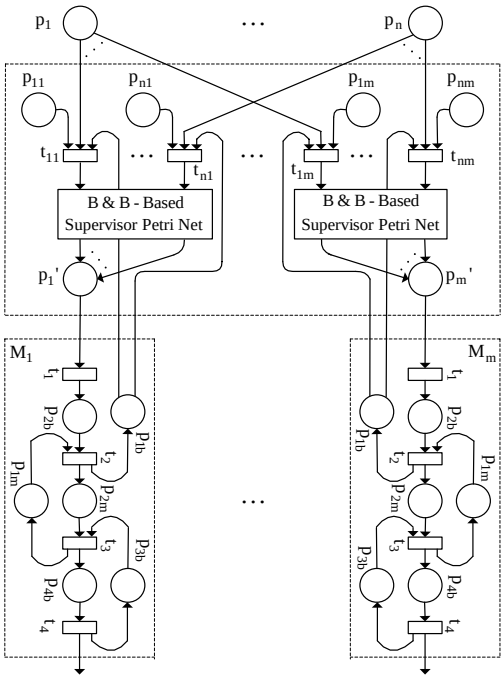

Fig. 5: Petri net-based application of the desired sequencing.

Because assigned jobs to each machine are sequence-dependent, we must put the token in the corresponding places in different instants and according to the sequence of the jobs scheduled on the machine. To solve this problem we introduce a B&B-based supervisor Petri net which enables us to put all tokens related to each machine simultaneously. Fig. 6 shows the Petri net model of the optimum sequencing by using B&B algorithm for n distinct jobs.

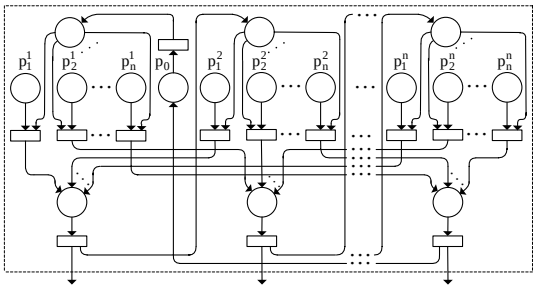

Fig. 6: B&B-based supervisor Petri net.

This Petri net considers different possible sequences of jobs by proper marking of places: $p_j^k \#\#k,\ j \in \{1,2,...,n\}$, where k indicates the ordering of jobs in the sequence and j represents the index of jobs. There isn't any restriction about simultaneous marking of the corresponding places. One of the places: $\{p_1^1, p_2^1, ..., p_n^1\}$ must be marked depending on the beginning job of the sequence. Marking one

place from each of sets: $\{p_1^2, p_2^2,..., p_n^2\}$, … and $\{p_1^n, p_2^n,..., p_n^n\}$ determines the next jobs of the sequence respectively. If we classify the places to n sets: $\{p_1^1, p_2^1,..., p_n^1\}$, $\{p_1^2, p_2^2,..., p_n^2\}$, … and $\{p_1^n, p_2^n,..., p_n^n\}$ or $\{p_1^1, p_1^2,..., p_1^n\}$, $\{p_2^1, p_2^2,..., p_2^n\}$, … and $\{p_n^1, p_n^2,..., p_n^n\}$, it can be seen that for each sequence only one place from each set is marked. In Fig. 6 this supervisor Petri net is used for supervision of the ordering of the jobs sequence assigned to each machine.

7. Computational Results

In order to evaluate performance of the heuristic approaches developed in this paper, a computational experiment was conducted. The results are provided in Table 1. In this computational experiment, for each testing problem, instances were generated and for each instance, the matrix of machining times was randomly generated. The minimum make-spans of the schedules obtained by the B&B were compared with the upper bound cost (UBC) by using $(\frac{\text{UBC}-\text{MC}}{\text{MC}})\times 100\%$. It can be seen from table 1 that in worst case the difference between UBC and MC is less than 100% of MC.

Table 1: Computational results

n	m	UBC	MC	$(\frac{\text{UBC}-\text{MC}}{\text{MC}})\times 100\%$
8	8	52	35	48.57
8	6	65	33	96.96
8	4	86	53	62.26
8	2	236	131	80.15
8	1	284	190	49.47
6	6	57	46	23.91
6	4	48	36	33.33
6	2	120	94	27.66
6	1	285	161	77.02
4	4	63	48	31.25
4	2	101	61	65.57
4	1	215	168	27.98

8. Conclusions

A new branch-and-bound algorithm was introduced to minimize the make-span of a parallel job-shop scheduling problem where it is assumed that the jobs are available at time zero and have sequence-dependent setup times on machines. In the developed new Branch and bound, nodes of the search tree were produced gradually and when the B&B algorithm needs to explore them. This causes that running the algorithm by computer doesn't need large size of memory. For modeling the manufacturing system and applying the desired sequence-dependent schedule on it a supervisor Petri net is introduced. The proposed methods were verified through a computational experiment.

9. References

[1] R. Z. Rios Mercado, and J. F. Bard. A Branch and Bound Algorithm for Permutation Flow Shops with Sequence Dependent Setup Times", IIE Transactions, (31): 721-731, 1999.

[2] S. Fujita, M. Masukawa, and S. Tagashira. A Fast Branch-and-Bound Algorithm with an Improved Lower Bound for Solving the Multiprocessor Scheduling Problem. Proc. of 9th international conference on parallel and distributed systems (ICPADS02), 2002, pp. 611-616.

[3] J. Jonsson, and K. G. Shin. A Parameterized Branch-and-Bound Strategy for Scheduling Precedence-Constrained Tasks on a Multiprocessor System. Proc. of International Conference on Parallel Processing, August 11-15, 1997, pp. 158-165.

[4] X. Wang, and J. Xie. Branch and bound algorithm for flexible flow shop with limited machine availability. *Asian Information-Science-Life*, 1(3), 2003.

[5] D. He, A. Babayan, and A. Kusiak. Scheduling manufacturing systems in an agile environment. Robotic and Computer Integrated Manufacturing, 17(1-2): 87-67, 2001.

[6] P. Y. Gan, K.S. Lee, and Y. F. Zhang. A Branch and Bound algorithm based process planning system for plastic injection mould bases. The international Journal of Advanced Manufacturing System, 18: 624-632, 2001.

[7] S. Olaffson, and L. Shi. A method for scheduling in parallel manufacturing systems with flexible resources. IIE Transactions, (32): 135-146, 2000.

[8] S. Khanmohammadi. Single array Branch and Bound method. Iranian Journal of Engineering, 3(1-2): 71-72, 1990.

[9] A. Jalilvand, and S. Khanmohammadi. Task scheduling in Manufacturing Systems Based on an Efficient Branch and Bound Algorithm. Proc. of IEEE Conference on Robotics, Automation and Mechatronics (RAM2004), December 2004, Singapore, pp. 271-276.

[10] A. Jalilvand, and S. Khanmohammadi. Modeling of Flexible Manufacturing Systems by Timed Petri Net. International Conference on Computational Intelligence (ICCI-2004), December 2004, Istanbul, Turkey, pp. 141-144.

[11] Tado Murata. Petri nets: properties, analysis and application. Proc. of IEEE, 77(4): 541-580, April 1989.

[12] A. Jalilvand, and S. Khanmohammadi. Using Petri Nets and Branch and Bound Algorithm for Modeling and Scheduling of a Flexible Manufacturing System. WSEAS Transaction on Systems, Issue 7, Vol.3, 2580-2585, Sept. 2004.

Enterprises Application Integration

Khubaib Ahmed Qureshi
SZABIST
Karachi, Pakistan
khubaib_ahmed@yahoo.com

Abstract

Business Integration (BI) has become a key issue for many companies to extend business market by integrating and streamlining processes both internally and with partners. To address this issue, whole marketplace has emerged for software solution that can help to achieve improved business integration which is referred as EAI. Originally EAI was only focused around integrating ERP with other applications within enterprise but now it is generally used as a catch-all term to cover all the other aspects of business integration. Major EAI approaches and evolution of enabling technologies ranging from EDI to Web Services and XML based process integration are analyzed to provide flexible, scalable and adaptable EAI framework. Solution comprises the challenge of efficiently integrating diverse business processes and data across the enterprises, allowing the organizations to keep pace with and respond to market changes.

1. Introduction

Due to the explosive growth of Internet, complex business expansion, competitive pressures, new business models, and the need to streamline business processes both internally and with partners or suppliers; has made BI key issue for many companies.

Enterprises need the level of internal and community based integration solution to help reduce cycle times, minimize the cost, and risk of connecting to entire value chain [1], increase response and outpace the competition. Every business need to constantly adapt and reconfigure their IT assets, systems, and business operations to meet changing customer demands; compress business cycles; and differentiate from competition. However, most enterprises have invested in packaged, legacy, and custom applications that perform specific business functions. Which operate within an extremely complex, inflexible, and mostly ad-hoc architecture consisting of monolithic silos, point-to-point connections and coupled hard coded interfaces. This makes it difficult to quickly assemble and reassemble the services, they provide as part of internal business processes or external business processes that support new and changing business requirements.

The ultimate goal of BI is therefore to have inter-and intra-enterprise applications evolve independently, yet allow them to effectively and conveniently use each other's functionality. Major challenge in business integration is interaction, which can be defined as consisting of interoperation and integration with both internal and external enterprise applications. Because enterprises applications are composed of autonomous, heterogeneous, and distributed components therefore offer challenges because of issues like scalability, volatility, autonomy, heterogeneity, and legacy systems. Real BI/EAI also requires conversion of varied data representations between partners' systems and connecting proprietary/legacy data sources, ERP, applications, processes, and workflows to the Web, and trading partners' systems [2].

Physiology is explained through continuum of EAI enabling technologies [3, 4], which has been used to solve business integration problem and classified as EDI based, component based message oriented middleware, workflows, XML framework, and Web Services. The study demands state of the art complementary EAI framework based on business process management harnessing the Event Driven & Service Oriented Architecture (EDA & SOA) [5].

Entire study of EAI approaches provide basis for proposed EAI framework that provides inter-and intra-enterprise wide process and data integration to enable real time business. It is recommended that real-time framework offer following Architectural features: (scalability, security, heterogeneity, adaptability, manageability, distributivity, decoupling, autonomy), real-time requirements: (asynchronous, publish/subscribe), and Business requirements: (flexibility, agility, usability, reliability) for productivity.

Purpose of this paper is twofold, first; highlight the