A DISTRIBUTED DISCRETE EVENT DYNAMIC MODEL FOR SUPPLY CHAIN OF BUSINESS ENTERPRISES

M. A. Jafari, and A. Amini

M. A. Jafari, Dep. of Ind. & Sys. Engineering, Rutgers University, +1-732-445-3627, jafari@rci.rutgers.edu
A. Amini, Dep. of Ind. & Sys. Eng., Rutgers University, +1-732-445-3602, amini@eden.rutgers.edu

Abstract—
We treat supply chain of business enterprises as a distributed system, where the system dynamics is governed by the events that occur within each enterprise and/or across several enterprises. While each enterprise has its own middleware for handling business transactions, we argue that there is also need for a meta-middleware across the whole supply chain. We present a formalism to build the control structure, which must be embedded in such a meta-middleware. We provide an illustrative example.

Keywords—
Business Enterprise, Discrete Event Dynamic System, Distributed Systems, Supply Chain.

I. INTRODUCTION AND PRELIMINARIES

A typical supply chain is a distributed system of manufacturing enterprises providing goods and services to the market place. With the advances in computer technology and Internet in late 1990s, the business collaboration amongst these enterprises shifted from EDI and fax based networking to B2B Internet based transaction processing. Many Internet based software solution packages have been developed and are currently being used which allow for an enterprise to connect over Internet or a VPN to its network of suppliers and to perform purchasing and order processing transactions. Generally speaking, each enterprise has its own enterprise resource planning (ERP), manufacturing execution system (MES) and other software solutions that internally handle its various business and other transactions. When it comes to the supply chain network, however, it is the collection of these, often, non-competitive systems that must work in collaboration and harmony in order to achieve the business objectives over the supply chain. Though, each enterprise operates over a hierarchical network of business and information layers, very often, the functions (such as actual production of goods at the manufacturing shop floor layer) becomes influenced by the decisions and tasks that must be performed in another enterprise (e.g., delivery of raw materials from suppliers).

It is quite common that the software systems and business protocols within an enterprise are designed and implemented in a stand-alone basis without taking into account potential business networking issues over a supply chain. In fact in many circumstances, networking with supply chain partners happens over time and can change by the market forces. It is also true that in the past few years, new technologies, such as Web Services have been developed allowing seamless interfacing between different business applications over the Internet. These services allow the business partners to focus on their specific business applications and avoid the development of APIs for low level software interfacing. We would like to categorize these types of technologies as pipelining technology, since they only provide the interfacing mechanism. *To our knowledge, there is still no technology and in fact no underlying model which can provide the means for collaboration between functions (we will toss the term "Agents") from one enterprise to work with agents from another enterprise within a supply chain.*

Our objective in this article is to treat *a supply chain of enterprises as a distributed system of logical agent (object) clusters where agents from one cluster (enterprise) can provoke methods or functions in*

the agents from another cluster. The dynamics of the individual enterprises and the overall supply chain is assumed to be event-based. Each cluster itself possesses a middleware, which provides all the necessary inter-agent pipelining and coordination of transactions and events within that cluster (enterprise). With our paradigm there must also be a *meta-middleware* to provide the same functionality between clusters. To better understand the functionality of the meta-middleware, we present a simple but an illustrative example.

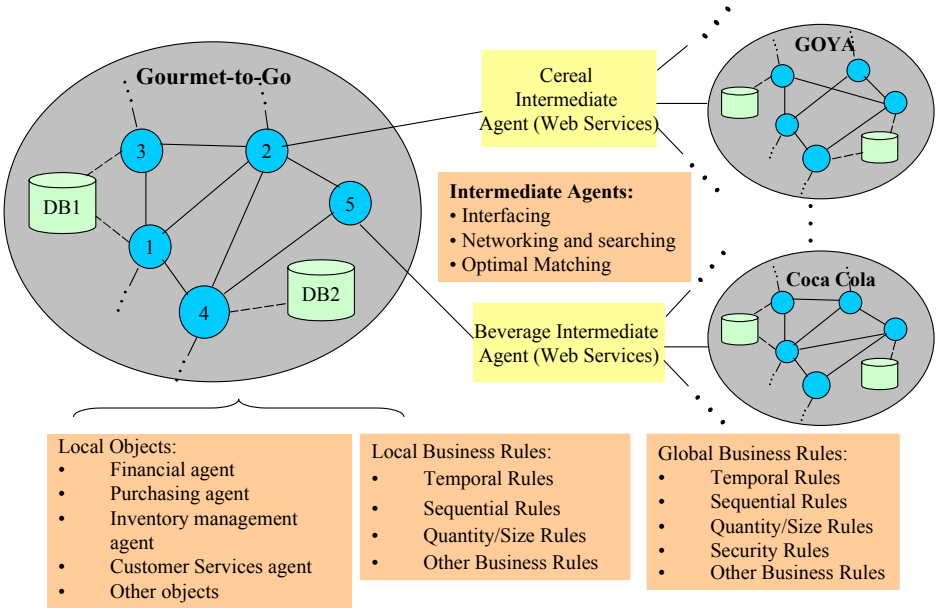

Fig. 1. Logical clustering of enterprises

Consider a Gourmet-to-Go restaurant, which provides catering services to households (Fig. 1). The customers make their orders over the Internet, by phone or in person. The restaurant in turn has its own suppliers providing it with raw materials, ingredients, beverages, etc. Logically speaking this supply chain can be viewed as a cluster of agents as shown in Fig. 1. The figure also gives sample objects or agents within a cluster together with intra-business and inter-business rules. Additional

Example Business Rules

Temporal: Check the inventory periodically *(local rule);*
Check the authorization of the intermediaries periodically *(global rule);*
Sequence: Update database following authorization from bank/credit card company *(local event);*
The Intermediary places order as soon as they find the first match *(global rule);*
The Intermediary must notify the Gourmet-to-Go before they place an order *(global rule);*
Quantity/Size: Orders with quantity > K must not be accepted by Gourmet-to-Go *(local rule);*
For any ingredient order > $C, the intermediary should notify Gourmet-to-Go first *(global rule);*

Example Events defined within the context of business process

Controllable: Database connectivity of agents *(local event);*
Select offers made by Web Services *(global event);*
Uncontrollable: Batch of meat on the supplier side is spoiled *(global event);*
Supplier initiates production *(global event);*
Customer places an order *(local event);*
Observable: Customer places an order *(local event);*
Network crashes *(global event);*
Unobservable: An Intermediary initiates a search *(global event);*
An Ingredient Supplier initiates production *(global event)*

Fig. 2. Examples of Business Rules and Business Events

examples of business rules and also possible events within the Gourmet-to-Go enterprise or between Gourmet-to-Go and its suppliers are listed in Fig. 2.

As it can be seen above, the events both local to an enterprise and global events can be categorized as controllable/uncontrollable or observable/unobservable. Controllability of an event implies that the event can be disabled/enabled or enforced by an agent controller (supervisory agent). Unobservability of an event implies that the event (local or global) may actually happen but not registered by the controller. These will be described later in more detail.

Generally speaking, the events occurring in one enterprise could potentially trigger activities in another enterprise within a supply chain. Each such an activity may in turn initiate events in that or other enterprises. So there are chain effects between the events occurring within the supply chain. Assuming that most or all of the functions at all layers of these enterprises are controlled by computers, what the meta-middleware will do is to monitor the sequence of these events and to ensure that the transactions initiated by these events are completed appropriately. The meta-middleware works across the supply chain and thus should take into account the business rules and events within each enterprise (perhaps in some aggregate level) and also the business rules and events between the enterprises (in detail level). What we are presenting in this article is the control structure that this middleware must adhere to in order to guarantee that business requirements specification across the supply chain is satisfied. More specifically, we present a methodology for synthesizing such a control structure. The *meta-controller* (imbedded in the meta-middleware) must be re-configurable in that if business requirements change within the supply chain, then the sequence of actions and events also change seamlessly. The meta-controller should also be scalable, in that, if additional enterprises, functions or agents were added to the supply chain, the meta-middleware seamlessly incorporates these into the overall system.

In terms of the current practices, what our methodology automatically provides is analogous to what the expert practitioners do manually using such tools as workflow or UML to design and coordinate business scenarios or to build database transactions. In terms of the new paradigm for software manufacturing, what we are proposing automates the business software assembly process where the RPC calls or interaction/linkage between object classes or software components are soft-wired according to the business rules and the underlying business processes. The soft-wiring can be changed *algorithmically* by the *meta-controller* if the underlying business processes or business rules change.

II. DESIGN OF THE META-MIDDLEWARE - A CONTROL THEORETIC APPROACH

Here we will focus our attention on the structure of the meta-controller, which will be embedded in the meta-middleware. We will start with a brief overview of the underlying theory. We will then present the approach using an illustrative example.

The control-theoretic approach to discrete event based systems (DES) was pioneered by Ramadge and Wonham [1]. Since then there has been many extensions of their work. Of particular interests are the work done in [2], [3]-[5], [6], [7] and [8]-[9], which extend their work to control synthesis for distributed systems and for system with imperfect information (e.g., some events are unobservable). Next we briefly describe some major concepts from this theory. A finite automaton (FA) is formally defined by a five-tuple $G = (Q, \Sigma, \delta, q_0, Q_m)$ where Q is the set of states, Σ is the finite set of alphabets or events, $\delta : \Sigma \times Q \to Q$ is the transition function, $q_0 \in Q$ is the initial state and $Q_m \subseteq Q$ is the set of marked (final) states. In this formalism each DES component is modeled as a finite automaton, G_i, with its own alphabet. In our application, each business component or agent is an FA with its own set of events. The overall or partial process automaton is then obtained by shuffling or parallel composition of the individual component automaton. Therefore, G, the finite automaton model of the

system is given by $G = \|_i G_i$, where $\|$ shows the shuffling (or parallel composition) operator. For the above illustrative example, G is the business process model of the Gourmet-to-Go.

The behavior of G is described by its language $L(G) = \{s : s \in \Sigma^* \text{ and } \delta(s, q_0)!\}$ where $\delta(s, q_0)!$ means $\delta(s, q_0)$ is defined, and Σ^* is the set of all possible strings constructed by any sequence of the members of Σ. For the example, a string in *L(G)* could represent a business transaction. Σ^* includes all strings of infinite size made from alphabet Σ. Marked language of G is specified by $L_m(G) = \{s : s \in L(G) \text{ and } \delta(s, q_0) \in Q_m\}$. For a business model, this language could possibly contain all the business transactions that result in some form of database operation. The events in Σ can be categorized by two disjoint controllable (Σ_c) and uncontrollable events (Σ_u), where the uncontrollable ones cannot be prevented or enforced. We note that controllability is defined in relation to a context. What this means is that an event controllable at a local level can be viewed as uncontrollable at one level higher. In the illustrative example, the event "Supplier initiates production" is uncontrollable to the Gourmet-to-Go while it is controllable to that specific supplier. The events in Σ can also be categorized by observability and unobservability. Again this is context dependent. For instance there may be many events in the supplier side that are unobservable to Gourmet-to-Go, but this does not mean that these events will not affect the Gourmet-to-Go operation. We will denote by $H = \{s \mid s \in \Sigma^*$ and s satisfies the specifications$\}$ the set of all transactions satisfying the specification (control requirements) provided by the business owner(s) or the market. From a practical perspective, this definition must be modified to reflect the business rules themselves and not those string or transactions that satisfy the business rules.

Given the shuffled finite automata model *G* of business process and business rules *H*, the next step will be to synthesize the control laws. We will use the concept of *a network of distributed controller agents* that will be in charge of executing the control laws over the network of businesses in a supply demand chain. ***Controller agents*** are capable of executing control laws defined by disabling, enabling, or enforcing controllable events on the business processes. Here we will demonstrate the synthesis of a single controller agent only. The implementation of this agent can be centralized or distributed.

From $L(G)$ and H we can synthesize a supervisor $\mathbf{S}$ using a standard algorithm developed by RW. This is on the assumption that all events are observable and the underlying process is deterministic. $\mathbf{S}$ has the following structure: $\mathbf{S} = (S, \Phi)$, where $S = (X, \Sigma, g, x_0, X_m)$ is a deterministic automaton with state set X, initial state x_0, a marked subset $X_m \subset X$, and transition function $g : \Sigma \times X \to X$ while $\Phi : \Sigma \times X \to \{1, 0, dc\}$ is a feedback function that selects a control policy $\Phi[x](\sigma)$ for each $x \in X$ and $\sigma \in \Sigma$. It can be shown that if supervisor $\mathbf{S}$ is used to control G, then the coupled language $L(\mathbf{S}/G)$ is the set of all strings that conform to H. Moreover the coupled language $L(\mathbf{S}/G)$ is the maximal set of such legal strings or business scenarios confining to H. Supervisor $\mathbf{S}$ also satisfies a number of other useful properties, such as non-blocking.

III. ILLUSTRATIVE EXAMPLE

Now we take a view of a subset of the processes that are going on in Gourmet-to-Go. Fig. 3 is a finite automaton of the customer service tasks of server i, the automaton that fills an order. The two sides of Fig. 3 are mirror images: the left side representing the filling of an order for fries and chicken; the right side representing the filling of an order for fries and beef.

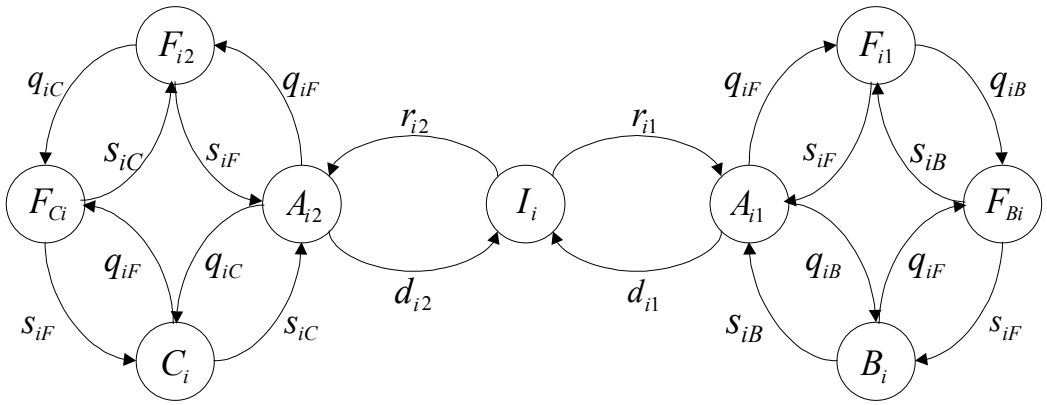

Fig. 3. Finite automaton of the customer service task of server I

This is an illustration of an enterprise having two products, each product having two components to be assembled. In the initial state (I_i), the server is idle. The arrival of an order for product 1 (r_{i1}) transitions the server to the state of preparing the order (A_{i1}). If a component of the assembly (fries, beef) is not available at this level of the production echelon, a requisition for replenishment is issued (q_{1F}, q_{1B}). When a requisitioned component arrives (S_{iF}, S_{iB}), the preparation of the order can continue (A_{i1}). Note that, in a complex manufacturing process, A_{i1} may represent the aggregation of a workflow of discrete tasks that includes production planning, scheduling, and other tasks that are part of fulfilling an order. It is not limited to just the physical assembly. There is another process going on in Gourmet-to-Go that is related to the server process. This is the requisition process, shown in Fig. 4.

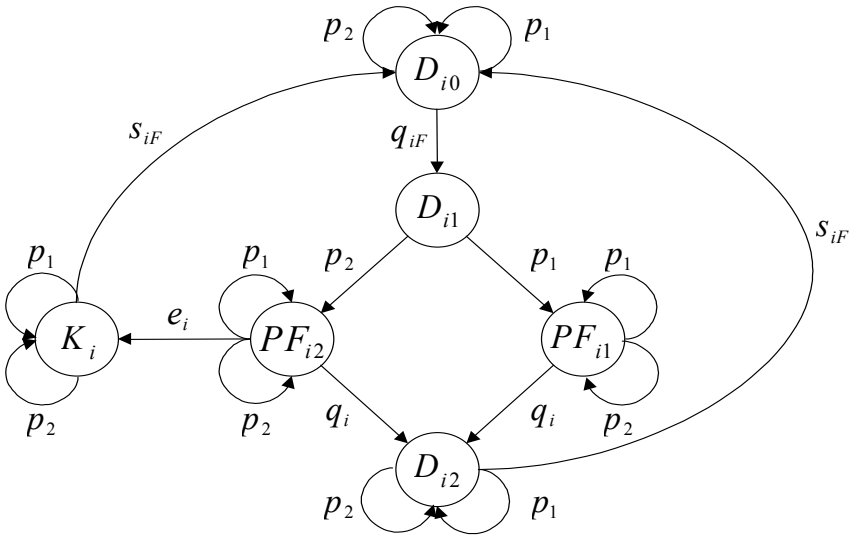

Fig. 4. Requisition process

Here, for illustrative purposes, we show only the requisition process for one component (fries). The requisition process for server i is initialized at D_{i0}, the state in which the server has no requisitions outstanding. The event q_{iF}, which also occurs in Fig. 3, transitions the requisition process to the state where a requisition is submitted (D_{i1}). The requisition is to be satisfied from a higher echelon of the Gourmet-to-Go inventory system. There are two inventory policies being modeled in Fig. 4. The state D_{i1} is the state where the requisition has been received and the state D_{i2} represents the state in which a purchase order has been processed. The right side transition set (p_1, q_i) shows a policy where, each time a requisition is received (D_{i1}), it naturally follows that a purchase order is released (D_{i2}). This is a lot-for-lot reorder policy, which implies that no inventory is being stored at this echelon. The left side transitions are a bit more complicated. There are two possibilities. The transition set (p_2, e_i) occurs

when the state of the inventory is $I \geq Q$; i.e., the inventory level is greater than the requisitioned quantity. These transitions result in state K_i, in which it is deemed that the requisition can be immediately satisfied from the existing inventory of this echelon. The transition set (p_2, q_i) occurs when the status of the inventory is $I<Q$; i.e., the inventory is less than the requisitioned quantity. This results in reaching the state D_{i2}; i.e., a purchase order is sent. Finally, the transition S_{iF} is the event that marks the completion of the delivery of the requisitioned material to the server. Note that this transition also appears in the server process of Fig. 3, where it returns that process to state A_{i1} and the server can now assemble the order. Thus, the server process and the requisition process are linked by common transitions (q_{iF}, S_{iF}). In a later section we will discuss the two policies modeled by the requisition process. It is important to note that other policies, such as a "base stock" policy or reorder point (Q,r) policy, can be easily modeled using this formalism. Here we limit ourselves to two simple policies for illustration.

The requisition process is related to another process, the purchasing process; it is the unfilled requisition that triggers the release of a purchase order. The purchasing finite automaton is shown in Fig. 5. From the initial state, P_{00}, the requisition of server i results in a purchase order being sent (q_i) and the purchasing finite automaton enters the state where an outstanding order for server i (i=1, 2) exists.

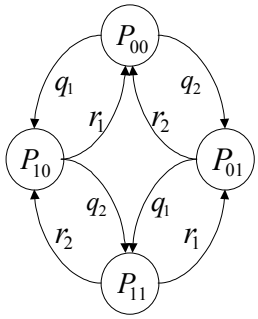

Fig. 5. Purchasing finite automaton

The subsequent delivery of that order is marked by the event r_i (i=1,2). Note that the purchase order process is related to the requisition process by the common transition q_i. The three finite automata described above show a simple example of the manner in which related processes in the "plant model" are linked through common events. In a manufacturing plant, server processes are typically shop floor activities and are considered part of the MES layer of the information system model of the plant. Requisition processes and purchasing processes are typically considered part of the ERP layer and the purchasing process itself is a link to other supply chain activities. All of these layers can be viewed as subsystems that are linked by common events. When all the relevant subsystem processes under consideration are appropriately specified, we refer to that collection of models as the "plant model". For our simplified illustration, the plant model consists of the server process, requisition process, and purchasing process.

Business Rules Models

Business rules should capture the logic of the order in which database transactions will occur during the execution of the plant model. In our formalism we model business rules as finite automata. Fig. 6 shows the specification of three such rules to govern our plant model. The finite automaton of Fig. 6a is initialized in state E_{i0}. It says that, in any instance of a workflow, a database transaction for e_i or r_i must occur before a database transaction S_{iF}. In other words, regardless of which policy is used in the requisition process (Fig. 4), there needs to be sufficient inventory (e_i) or a new order must arrive (r_i) before a re-supply to the server echelon is made (S_{iF}). In any instance of a workflow, if a transaction S_{iF} occurs prior to e_i or r_i, an error in the process is deemed to have occurred or a database transaction that should have been observed was not observed.

In a similar manner, Fig. 6b illustrates another rule: a requisition must be processed before the relevant requisition policy is executed. The "relevant policy" is a decision variable that may be selected by management. In Fig. 6c, state J1 is the state in which requisition processing policy 1 (right-hand side of Fig. 4) is in use. State J2 is requisition policy 2 (left-hand side). The transition "n" is a management decision event. Fig. 6 illustrates three business rules: two sequential rules (Fig. 6a,b) and one decision rule (Fig. 6c). In our formalism, these rules are defined and/or changed separately from the underlying processes of the plant model. In a final step the business rules are combined (coupled) to the enterprise model in order to constrain the evolution of the processes within the enterprise to behave according to these business rules. The result is shown next.

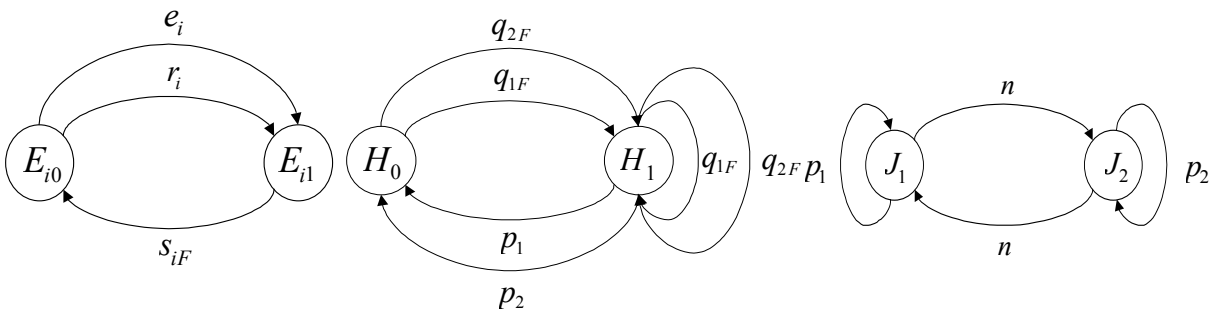

Fig. 6a Fig. 6b Fig. 6c

Fig. 6. Business Rules

Resulting Workflow

When the set of business rules are coupled to the enterprise model, the resulting digraph shows the possible ways in which the events in the enterprise can evolve within the constraints of the business rules.

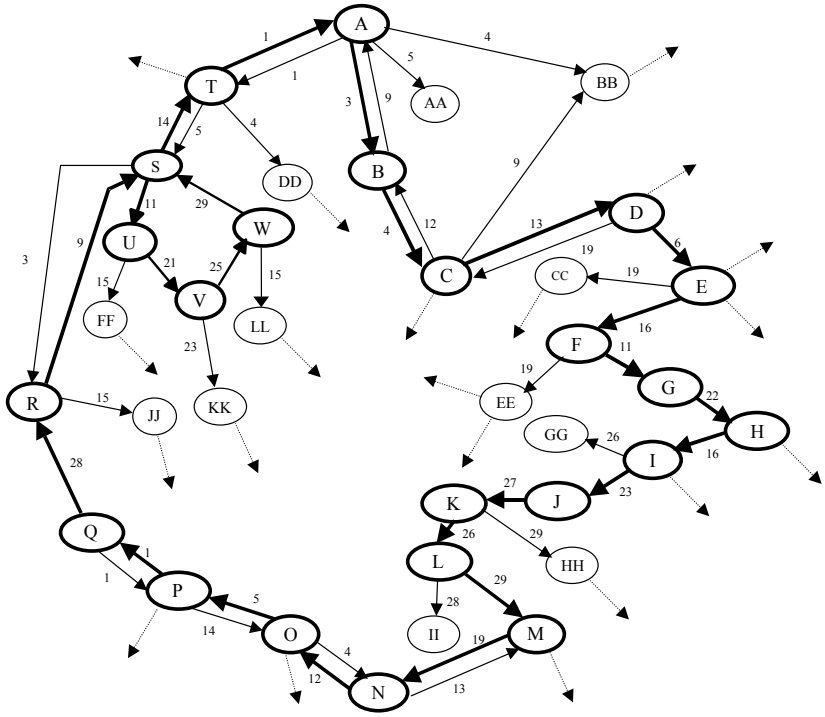

Fig. 7. Business scenarios (partial set)

We refer to this as the "business scenarios", a partial set of which is shown in Fig. 7. These scenarios are automatically obtained. State A is the Cartesian product of the initial states of the finite

automata of Fig. 3 - Fig. 6. Any path from state A back to state A is a circuit that represents a legal business scenario; i.e., it conforms to the business rules. The digraph of the business scenarios is shown as a generic model in Fig. 7.

Any particular instance of a circuit that is initiated at state A with the event r_{ij} is defined as a legal sequence of business transactions. Two such sequences, extracted from Fig. 7, are shown in Fig. 8. These sequences are shown here using workflow symbolism, but other symbology can also be used. In Fig. 8, server 2 processes two orders, running out of fries each time. In the first sequence, policy 1 is in use. The sequence terminates in state 8. Note that the path generated by the digraph is legal in the sense that the constraints of Fig. 6a and 6b hold. Our formalism will generate only legal paths. The second workflow begins when r_{22} transitions into state 9. Coincidentally, management is changing one of its business rules; i.e., it switches to Policy 2. The resulting sequence again follows the constraints of Fig. 6.

The legal workflow diagram shown in Fig. 8 is generated automatically from knowledge of the enterprise operation coupled with the business rules (Fig. 6). The resulting process is dynamic; i.e., business rule or policy changes can be automatically reflected in a new legal workflow.

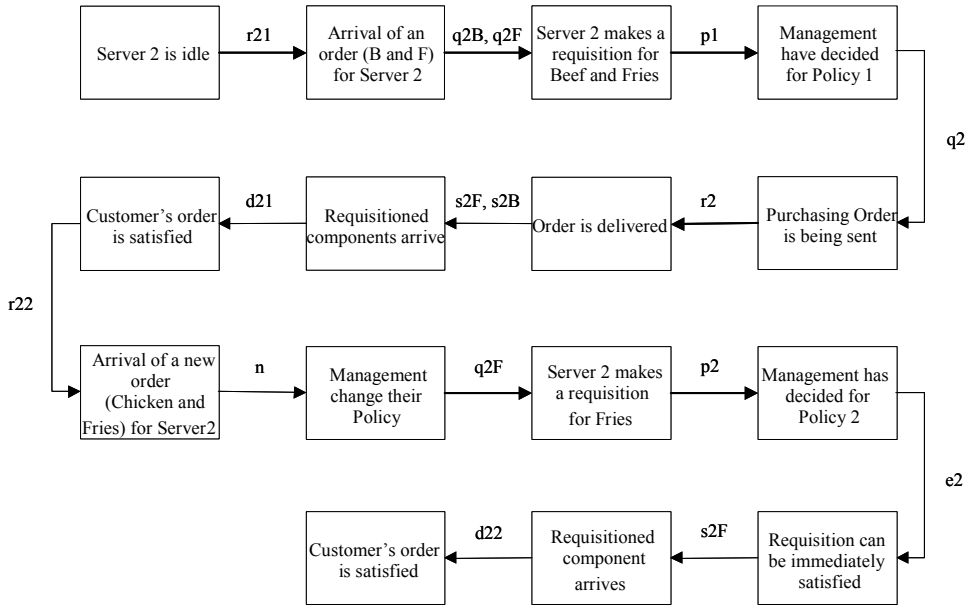

Fig. 8. Legal workflow extracted from business scenario

Analysis of Workflow Properties

We used the Finite Automata analysis techniques introduced by Ramadge and Wonham [1], and Darabi and Jafari [8]-[9] to investigate certain properties of the enterprise supply chain. Table I provides the list of these techniques and their related properties. In the following we describe some applications of these techniques for our Gourmet-to-Go restaurant supply chain.

When the set of business rules are coupled to the behavioral model by RW algorithm, the resulting workflows show the possible ways in which the business transactions can evolve within the constraints of the business rules. These are referred to as "business scenarios", a partial set of which is shown in the following Fig. 8 (The complete set for this case study has 2100 states, which is not shown here). In the first sequence, policy 1 is in use. The sequence terminates in the 8th state starting from the initial state. It is noted that the path generated by the digraph is legal in the sense that the constraints of Fig. 6.a, 6.b and 6.c hold. The second sequence begins when r_{22} transitions into the 9th state.

Coincidentally, management is changing one of its business rules; i.e., it switches to policy 2 (stock replenishment). The resulting sequence again follows the business rules. The resulting process is dynamic; i.e., business rule or policy changes can be automatically reflected in a new legal workflow.

TABLE I
SUPPLY CHAIN ANALYSIS TECHNIQUES AND PROPERTIES

Analysis Technique	Properties
Consistency checking and legal workflow existence	-Verifying if the current supply chain routines can satisfy all business rules, constraints and requirements. -Creating a supply chain workflow that is used to co-ordinate the supply chain activities and their secure execution (meeting the business rules).
Control specification redundancy	-Verifying if the current business rules are redundant -Analyzing the supply chain reaction to the introduction of a new business rule (or supply chain constraint).
Event redundancy	-Identifying the necessary information for supply chain coordination decisions. -Identifying the information that is generated but not necessary for the control of the supply chain.
Event controllability analysis	-Investigating the reaction of supply chain to the addition of extra control or removal of current control means.
Task scalability	-Automatic updating of supply chain execution procedures, when a new task is introduced or a current one is removed.
Specification scalability	-Developing a parametric formulation of business rules. For example, verifying how an inventory system execution reacts to different number of orders.

The properties presented in Table I can be investigated for this supply chain. For example, it can be shown that the delivery event d_{11} in Fig. 3 may be unobservable. In other words there is no need to receive the information on the occurrence of this event to guarantee the conformance of the restaurant activities with its business rules.

Implementation and Performance Analysis

We developed a software system to implement the modeling and analysis framework discussed above. Fig. 9 shows the software system architecture. As one can see, in addition to DES control properties, we also provide performance optimization of the controlled supply chain.

The input module provides a graphical interface to the user to model different enterprises of the supply chain, their activities and their related business rules. In addition to the state transition data of the supply chain tasks, the users also enter tasks execution times. The current software allows only deterministic and exponential time durations for tasks. Two modules “RW algorithm” and “Normal Supervisors Algorithm” have implemented all the techniques listed in Table I. RW module receives the input data and it constructs the RW controller of the supply chain. Normal Supervisors module uses the control structure to make all normal partial observation policies of the supply chain. The task duration times and the partial observation policies are fed to a Markov Decision Process (MDP) to find the best observation policy under different supply chain conditions. Linear Programming is used to find the optimal solution of the MDP. This module also provides the rules to switch between different

partial observation policies. The switching rules can define the reconfiguration policies of a supply chain when its information set changes over time. Specifically, the reconfiguration policies can be applied to guarantee the proper execution of the supply chain proper when missing database fields or failed communication messages are encountered. The Output module generates different reports and displays of the supply chain modules, their execution policies, their performance and the reconfiguration map.

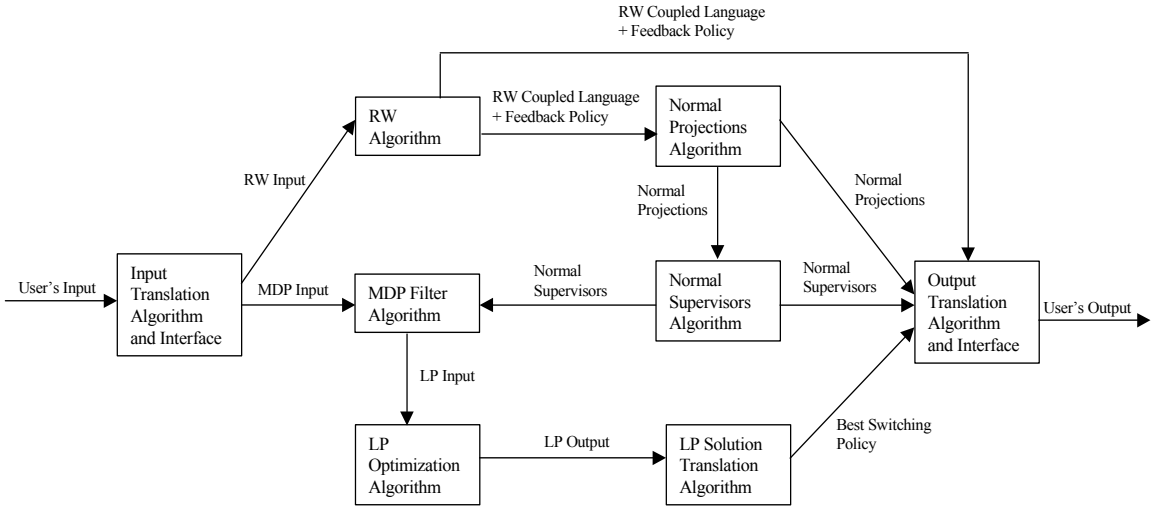

Fig. 9: Software Architecture

IV. SUMMARY

We treat supply chain of enterprises as distributed systems, each enterprise being a cluster of logical agents. The agents from different clusters (enterprises) are inter-linked. The dynamics of this system is event based. While each enterprise possesses its own middleware to coordinate, monitor and control the internal business transactions, we argue that a meta-middleware at the supply chain level is also needed. This middleware in essence coordinates and controls the transactions taking place between the enterprises within the supply chain. We present a formalism to automatically build the control structure which must be embedded in such a middleware. The model obtained from our metrology defines how the inter- or intra-cluster agents must be soft-wired together. The soft-wiring allows for dynamic change of the control structure as enterprise processes or business rules change due to market forces or other reasons.

REFERENCES

[1] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event Processes," *SIAM Journal on Control Optimization*, vol. 25, no.1, 1987, pp. 206-230.

[2] R. Cieslak, C. Desclaux, A. S. Fawaz, and P. Varaiya, "Supervisory control of discrete-event processes with partial observations," *IEEE Transactions on Automatic Control*, vol. 33, no. 3, 1988, pp. 249-260.

[3] F. Lin, and W. M. Wonham, "Decentralized control and coordination of discrete-event systems with partial observation," *IEEE Transactions on Automatic Control*, vol. 35, no. 12, 1997, pp. 1330-1337.

[4] F. Lin, and W. M. Wonham, "Supervisory control of timed discrete-event systems under partial observation," *IEEE Transactions on Automatic Control*, vol. 40, no. 3, 1995, pp. 558-562.

[5] F. Lin, and W. M. Wonham, "On observability of discrete-event systems," *Information Science*, vol. 44, no. 3, 1988, pp. 173-198.

[6] T. Ushio, "On the existence of finite-state supervisors under partial observations," *IEEE Transactions on Automatic Control*, vol. 42, no. 11, 1997, pp. 1577-1581.
[7] R. Kumar, and M. A. Shayman, "Centralized and decentralized supervisory control of nondeterministic systems under partial observation," *SIAM Journal On Control Optimization*, vol. 35, no.2., March 1997, pp.363-383.
[8] H. Darabi, and M. A. Jafari, "Optimal supervisory control switching under sensory failures," *IEEE Transactions on Robotics and Automation*, Dec. 1999, submitted for publication.
[9] H. Darabi and M. A. Jafari, "Recovery analysis of supervisory control of discrete event systems," *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, San Diego, CA, Oct. 1998.