# Autonomy in Collaborative Manufacturing Networks

Yun Guo and Amihai Motro
Department of Computer Science
George Mason University
Fairfax, VA
{yguo7, ami}@gmu.edu

***Abstract*—A collaborative manufacturing network is an alliance of business entities (mostly manufacturers and suppliers) who collaborate on the production of complex products. An essential element of this type of collaboration is that the participating entities are allowed to retain some measure of *autonomy*. In this paper we examine different approaches to collaborative manufacturing that support different levels of cooperation and autonomy. We identify three major architectures for collaborative manufacturing networks, and we describe a design for a single platform that supports all three architectures. The platform is based on three major components: an information repository, a service repository and a workflow repository. We also address several challenging issues, such as collaboration failures, management of supply capacities and order quantities, and risk estimation.**

## I. Introduction

The term *collaborative network* describes a coalition of partners who integrate their capabilities and resources to achieve common goals. The operation of collaborative networks is typically supported by computer networks, allowing the participants to be geographically distributed. The collaborative network paradigm has been deployed in a variety of disciplines, including learning, health care and information retrieval. When the common goal concerns *business opportunities*, the collaborative network is often referred to as a *virtual enterprise*. An example of a virtual enterprise is an alliance of manufacturers and suppliers who collaborate on the production of complex products — in which case we refer to it as a *collaborative manufacturing network*.[1]

An essential element of this type of collaboration is that the participating entities are allowed to retain some measure of *autonomy*. That is, each participant is an independent business entity that, while participating in the collaborative network, continues to maintain a degree of independence. This manifests itself in different ways: While satisfying the common goals, the participant may pursue additional goals, protect its information resources, or participate in multiple collaborative networks. Consequently, the different architectures that have been proposed for virtual enterprises exhibit quite different approaches to autonomy.

In this paper we examine different approaches to collaborative manufacturing that support different levels of cooperation and autonomy. We observe four fundamental feautures with which the different approaches may be distinguished, and we combine these feautures to create three major types of virtual enterprises, which we call *centralized*, *distributed* and *autonomous*. Section III describes our underlying model for virtual enterprises, and identifies these three district architectures.

Realizing that no particular approach is the "best", we advocate the creation of a single platform for launching and running all three types of virtual enterprises. The common platform consists of three major components: an *information repository* to support the data needs of the enterprise, a *service repository* to provide the basic functionalities, and a *workflow repository* to describe the different workflows that weave together services and information from the two aforementioned repositories. These components are designed to be shared among the different architectures wherever possible. Sections IV, V and VI describe these three major components. A small example is then presented in Section VII to illustrate the principal differences among the three architectures under consideration in this paper.

The workflows we describe involve several challenging issues including: collaboration failures (e.g., participants quitting or not responding), risk estimation (to assist both clients and participants in making procurement decisions), and cached optimizations (that by the time executed might no longer be optimal). Preliminary treatment of these issues is provided in Section VIII. Finally, Section IX summarizes the contributions of this paper and outlines future research directions. We begin in Section II with a brief survey of related work.

## II. Background

Work related to this paper can be divided into three groups: models, issues and platforms.

### A. Models

It seems appropriate to begin this survey with research on model development for collaborative networks. VirtuE [1] is a formal model for describing collaborative manufacturing. The focus is on information products, but the model can be applied in other disciplines. Among other features, it introduces enterprise rules to enforce enterprise policies, and computed indicators to track performance performance. ARCON [2] proposes a comprehensive modeling framework that considers several dimensions (termed structural, componential,

[1]Although a collaborative manufacturing network is an example of a virtual enterprise, we shall use the terms interchangeably. That is, the virtual enterprises that we discuss focus on manufacturing.



978-1-63190-043-3 © 2014 ICST
DOI 10.4108/icst.collaboratecom.2014.257369

functional and behavioral) across the life cycle of an enterprise. Collaborative network modeling can also be based on mature enterprise modeling frameworks such as the framework suggested by Zachman [3] or the Open Group Architecture Framework (TOGAF) [4]. The model defined in this paper follows the overall approach of VirtuE in that it is built around a small set of formal concepts, and collaboration is regulated in specific procedures. This approach allows formal treatment of issues such as risk, failure and optimization. Most importantly, the model allows architecting systems with different levels of participation and autonomy. Additionally, whereas enterprise modeling frameworks provide a generic framework for any type of enterprise, this model focuses on manufacturing collaborative networks.

Our manufacturing model borrows concepts from supply chain network (SCN). As in SCN, our model guides the strategic decisions of the participants, aiming to satisfy client orders, while achieving optimized value. SCN design models can be classified into deterministic [5], [6] and nondeterministic [7], [8], [9] depending on the use of stochastic parameters. Our manufacturing model is a simplified static-deterministic model derived from OptiVE [10]. In this model, product types are either elementary or complex, the capacity of elementary products is predetermined, production is in multiple stages, and various product properties are aggregated in formulating optimal production goals. As with most static deterministic SCN design models that can be formulated as mixed integer programming (MIP) [11], [12] optimality can be achieved efficiently with recent versions of commercial solvers such as CPLEX [13] and Xpress-MP [14].

### B. Issues

Several issues that are considered in this paper have been addressed by others, albeit not in the same context. Using concepts from coalitional game theory [15] addresses the issue of enterprise *formation*: how to determine the membership of virtual organization to reduce cost and guarantee maximum profit. The dual phase of enterprise *dissolution* is addressed in [16], which studies the different causes that justify dissolution. Our interest in tracking and advertising the performance the participating entities is strongly related to the concept of *reputation* investigated in [17].

### C. Platforms

In the *service-oriented* programming paradigm, programs are composed by weaving together platform-independent software components that are accessible over the World Wide Web [18]. The deployment of this paradigm for architecting virtual enterprises has been suggested in several works. Usually this paradigm is used to model enterprise *participants* as service providers [19], [20], [21]. Our approach here is different: We use services to provide the different *functionalities* of the system (e.g., messaging, data management and optimization). We argue that this paradigm is well suited for the requirements of collaborative network. It facilitates sharing data and services among geographically distributed participants; it alleviates the compatibility issue among participants of heterogeneous organizations; and it enables rapid retrofitting of enterprises to a new style or architecture.

An alternative paradigm has been suggested that uses an *agent-based* approach. In [22], agents are used to represent the candidate entities in the process that determines who participates in a virtual enterprise. More generally [23] describes a platform that uses agents to support most of the enterprise functionalities throughout the life-cycle of the enterprise (creation, operation, evolution and dissolution).

## III. Architecture Classification

We begin with a description of the fundamental assumptions of our model for collaborative manufacturing networks. These assumptions allow architecting virtual enterprises of many different types. We then identify four major feautures with which different architectures may be distinguished. By choosing different combinations of these feautures, six different architectures may be created. Among these, we focus on three highly distinct architectures (the other three may be considered small variations of these).

### A. Fundamental Concepts

Our model of virtual enterprises adopts these basic concepts and terminology.

**Marketplace and members.** The marketplace is a collection of business entities that are registered as willing to participate in collaborative manufacturing. These business entities are referred to as *members* of the marketplace. Members of the marketplace have properties such as *reliability* (an accumulating score of performance).

**Virtual enterprises.** A virtual enterprise is a group of marketplace members that have agreed to collaborate on the manufacturing of specific products. Typically, these members possess complementary skills and resources that are necessary to manufacture these products. A participating member is referred to as an *affiliate* of the enterprise. The affiliate's unit is called a *division*. A member of the marketplace could be affiliated with different virtual enterprises.

**Products and versions.** Each affiliate is capable of manufacturing certain products from "component" products that it procures from other affiliates, and then making its products available to other affiliates as components for their products. The ultimate goal of an enterprise is to manufacture "end" products that are ordered by outside clients. For successful exchanges of products among affiliates, products are identified in a *catalog* shared by the entire marketplace.

A specific implementation of a product by an affiliate is referred to as a *version* of the product.[2] Product versions are either elementary or complex. An *elementary* product version is manufactured by an affiliate without the need to procure any components from other affiliates. A *complex* product version is manufactured from other products, either

[2]Products are analogous to *types* and versions are *instances* of these types.

complex or elementary, that must be procured from other affiliates. If the product is complex, then the affiliate has a *plan* for manufacturing it. The plan is simply a list of necessary component products and their quantities. Although entirely interchangeable, different versions of a product could have different plans. Indeed, an affiliate could make available different versions of the same product, each with a different plan.

**Properties of products.** Each product version offered by an affiliate is associated with various properties. Examples of properties include the *price* that must be paid by any affiliate that wishes to procure it, the *cost* of manufacturing the product version, the *complexity* of the version, which is the number of elementary products embedded in it (it is the sum of the complexities of all the product versions used in manufacturing it), the promised *time* to delivery, and the *risk* that the product would not be delivered on time (or not at all).

**Production workflow.** Typically, an affiliate receives an *order* to manufacture a product version. It then issues orders to other affiliates for the component product versions in its plan. When these orders have been fulfilled, it manufactures the product version that has been ordered from it, and delivers it to the appropriate affiliate. This results in a workflow in which orders are propagated across the network, and eventually fulfillments are propagated in the reverse direction. The ordering phase just described may be preceded by a *quoting* phase in which quote requests are propagated and answered.

### B. Distinguishing Feautures

In the environment set up by these fundamental concepts and assumptions, enterprises may be architected in different ways to provide different levels of autonomy. To classify these different architectures, we identify four feautures with which the different architectures may be distinguished. These four feautures represent a first attempt at the classification of autonomy, and admittedly, there could be additional feautures. Moreover, our feautures are all binary, where possibly there could be more elaborate architectures where some feautures would possess "intermediate" values.

**Does the enterprise have a leader?** We distinguish between *centralized* organization, in which the virtual enterprise has a *leader*, and a *distributed* organization in which the virtual enterprise is *leaderless*.

**Who controls ordering?** In every virtual enterprise, the manufacturing affiliates determine the *plans* for manufacturing; that is, the parts and the quantities that are needed to manufacture a single product. But enterprises could differ as to who selects the suppliers and quantities that will be used to fulfill orders. These decisions could be done *locally*, by each affiliate, or *globally*, by the leader of the virtual organization.

**When are orders optimized?** Selecting the suppliers and the quantities to fulfill orders that have been received requires optimization. Such optimization could be performed *on-demand*; that is, when the order is received, or *in-advance*; that is, unrelated to a particular order.

**How many phases in the ordering workflow?** The workflow of processing orders may involve one or two phases. In a one-phase workflow orders are propagated and products are returned. In a two-phase workflow, this phase is preceded by a *quoting* phase.

These four binary choices can be combined to create 16 architectures (or *modes of operations*). We denote each architecture with 4 characters:

1) Organization: C (centralized) or D (distributed)
2) Control: G (global) or L (local)
3) Optimization: O (on-demand) or I (in-advance)
4) Phases: 1 (one-phase) or 2 (two-phase)

For example, CGO1 denotes an architecture that is centralized (with a leader), the leader determines the details of the entire supply chain, these decisions are made when orders are received, and the workflow of processing orders does not involve quoting.

### C. Three architectures

Upon close examination, ten of the 16 combinations are of little or no interest. First, since a distributed architecture has no leader, control of ordering is always with the affiliates (i.e., D $\Rightarrow$ *L*); hence the four combinations DG** make no sense. Second, when orders are processed in two phases, optimization cannot be done in-advance, only after quotes have been received (i.e., I $\Rightarrow$ 1); hence the combinations **I2 make no sense, either. Third, combinations of the type *LO1 are also problematic, as they indicate that each affiliate optimizes its orders when they are received. This requires that the affiliate has prices from its suppliers, but these suppliers cannot provide prices until *they* receive orders. Finally, the combination CGO2 is pointless: Since this is a centralized enterprise with global control, two phases are never required, because with access to the data of the affiliates, the leader could discover what would be the best quotes.

This leaves only six viable combinations: CGO1, CGI1, DLO2, CLO2, CLI1, and DLI1.

Of these combinations, the second, fourth and sixth can be viewed as small variations of the first, third and fifth, correspondingly. In each pair, the difference is in a single feauture, which results in architectures that are not radically different. In this paper we focus on three combinations (one of each pair), with resulting architectures that are substantially different from each other:

| Model | Organization | Control | Optimization | Phases |
|---|---|---|---|---|
| CGO1 | Centralized | Global | On-demand | One |
| DLO2 | Distributed | Local | On-demand | Two |
| CLI1 | Centralized | Local | In-advance | One |

We shall refer to the three selected architectures, as *centralized* (CGO1), *distributed* (DLO2) and *autonomous* (CLI1). In previous papers, we described a centralized architecture called OptiVE [10] and an autonomous architecture called SOAVE [24].

Recall that our goal is to create a single platform that will support all three architectures. Users of the platform will

be able to indicate the type of virtual enterprise that should be launched. After creation, the enterprise would operate in the manner consistent with the chosen architecture. Our platform evolves around three major components: an *information repository* to support the data needs of the enterprise, a *service repository* to provide the basic functionalities, and a *workflow repository* to describe the different workflows that weave together services and information. These components are described in more detail in the following three sections.

## IV. Information Repository

The information repository maintains the data that support the operation of the virtual enterprise. It uses the relational data model to store data in tables. The tables are of three types based on their accessibility: *external* tables store information that is available outside the enterprise (for example, to potential clients); *global* tables store information that is available to all the affiliates of the enterprise; *local* tables store information that is available only to individual affiliates of the enterprise.[3]

There are only two external tables. *E_Marketplace* is a registry of the community of members that are available for participation in virtual enterprises. For each member, it provides contact information, manufacturing capabilities and performance statistics. *E_Public* is a catalog of the different product versions that available for ordering from the enterprise, along with their essential properties (e.g., the price, the time to delivery, or the risk that the product will not be delivered as promised).

The global and local tables are organized in two layers: a *common* layer of tables that are used in every architecture, and a layer of three individual sets of tables, each used in a particular architecture only.

The common layer includes three global tables and one local table. *G_Catalog* describes the products of the enterprise, both end products to be offered to external clients, and component products to be used by affiliates in their manufacturing plans. *G_Directory* describes the affiliates of the enterprise (it is a subset of *E_Marketplace*). *G_Order* is a complete log of the orders of product versions executed by the enterprise. The subset of orders pertaining to an individual affiliate is stored in *L_Order*. Each log entry includes about a dozen fields, including the time order was received, the status of the order, the time it was delivered, the quantity ordered, and the price.

In the autonomous architecture, the four common tables are supplemented with three tables: *L_Availability* describes the product versions offered by each individual affiliate, including fields such as price, time-to-delivery and risk. *G_Availability* is the union of the individual local availability tables (the aforementioned *E_Public* is a view of *G_Availability* showing only versions of end products). *L_Plan* stores the bill-of-materials information for each product version offered by an affiliate.

The centralized architecture uses three similar tables, but with somewhat different attributes. However, access to the local tables is granted to the enterprise leader. The distributed architecture uses the same three tables as the centralized architecture, and adds *G_Request* and *G_Proposal* to track the quoting process.

Altogether, the information repository supports the three architectures with a total of 14 tables and views.

[3]In the following, the three types are denoted with prefixes *E_*, *G_* and *L_*.

## V. Service Repository

A service-oriented software approach is an attractive solution for a platform that is intended to support different architectures. This modular approach facilitates sharing of functionalities among the different architectures. A repository is established to house services that are accessible to the affiliates of the enterprise. Repository services are of three types: *management*, *production* and *information and communication*,

Three management services are provided to control the community of the enterprise, its evolution, and its offerings. To manage the enterprise community, a *membership* service provides functions for inviting and dismissing affiliates, and keeping track of their performance scores. Note that in a distributed architecture members join and leave an enterprise on their own. An *evolution* service provides functions for creating and dismantling enterprises, and for creating or dismantling enterprise divisions (these are invoked automatically when an affiliate joins or departs). In a distributed architecture, creation and dismantling of an enterprise is invoked automatically as the first affiliate joins or the last affiliate departs. The offerings of a virtual enterprise are managed by a *resources* service. It enables adding, removing and modifying products, product versions, and plans.

Three production services support the manufacturing process. An *optimization* service optimizes the procurement process. It is invoked with a product and a desired quantity, locates the available plans throughout the enterprise, and based on the objectives of the initiating affiliate, calculates the best procurement options for producing the target product in the specified quantity. In locating component products the optimization service considers product parameters such as price, risk, and time-to-delivery. The objectives of the initiating affiliate could involve any of these parameters, and possibly a weighted combination of parameters. The optimizer could possibly fail, when it cannot not construct any plan for the given product and quantity. In the autonomous and distributed architectures, where procurement decisions are determined by each affiliate, the optimization service is invoked by individual affiliates. The service then returns the best procurement strategy for the initiating affiliate. In the centralized architecture, where all procurement decisions are controlled by the leader (it has access to the production plans of all the affiliates), the service is invoked by the leader. The service then returns the best procurement strategy for the entire enterprise.

At the completion of an order initiated by a client a *performance tracking* service is invoked. It examines final values in the global and local order logs to determine whether individual affiliates performed as promised. The output is then used to adjust the reliability scores of the affiliates. The distributed

architecture requires an additional service to manage *quote requests*. This service distributes requests for quotes on behalf of an affiliate, collects the responses from other affiliates, and delivers them to the requesting affiliate.

There are two services in the information and communication group. Virtually all the activities carried out by the above mentioned services require access to the information repository. The management of this repository is performed exclusively by an *information* service. Finally, a *messaging* service facilitates communication among the affiliates of the enterprise. Enterprise affiliates send messages to each to execute transactions and to perform management functions. These include requests for price quotes, responses to quote requests, orders, deliveries, invitations to join the enterprise, dismissals from the enterprise, and so on.

## VI. Workflow Repository

Business process workflows of virtual enterprises are collections of activities scheduled according to specific business logics. In building business process workflows, services from the service repository serve as building blocks. By assembling these blocks in customized workflows, virtual enterprises may be devised to operate in different modes. This design not only reduces development time by reusing existing functionalities, but also supports flexibility and diversity of virtual enterprises. Workflows are stored in a repository, allowing modification, addition of new workflows, or deletion of exiting workflows.

Virtual enterprises employ different workflows; for example, to launch an enterprise, to add a product, to terminate an affiliate, and so on. Of these, the most critical is perhaps the *production workflow* that defines how collaborative manufacturing is performed. Since more than anything else, these production workflows convey the individual style of each architecture, we sketch here the production workflows of each of the three architectures.

**Autonomous architecture.** This architecture is centralized with local control. The leader launches the enterprise, determines its products, invites affiliates, and manages all external orders. But the affiliates operate autonomously: They determine the enterprise products they wish to manufacture, they choose the suppliers and quantities for these products, and they set their own prices. The suppliers and quantities are determined in advance, so the execution of transactions is efficient. The downside, however, is that price or availability updates by suppliers may require re-optimization.

1) A client consults the $E\_Public$ catalog for the available product versions and their essential parameters and submits an order to the leader, specifying a version and a quantity.
2) After verifying the validity of the order, the leader acknowledges the order with an order number.
3) The leader forwards the order to the manufacturing affiliate (each version is offered by a specific affiliate).
4) The affiliate launches a production:
    a) It consults its $L\_Plan$ table and sends its suppliers the appropriate orders.
    b) When orders have been fulfilled, the affiliate assembles the product and delivers it to the ordering affiliate.
    c) The ordering affiliate acknowledges with payment and proceeds to assemble its own product.
5) These steps repeat until the leader receives the final product and responds with payment.
6) The leader sends the product to the client, who responds with payment.

**Centralized architecture.** This architecture is centralized with global control. In addition to the functions described in the autonomous architecture, the leader exercises control over the entire production process. Having access to the manufacturing capabilities of each affiliate, it generates a global production plan and instructs its affiliates on their choice of suppliers and quantities. Indeed, affiliates are complete subordinates that put their manufacturing capabilities at the disposal of the leader.

1) A client consults the $E\_Public$ catalog for the available products and the offering affiliates and submits an order to the leader, specifying a product and a quantity.
2) The leader forwards the order information to the optimization service. The service scans the global $G\_Availability$ and the local $L\_Plan$ tables to compute an optimal production plan for the order. It sends this plan to the leader.
3) The leader launches a production. It sends each affiliate named in the plan specific instructions:
    a) What to manufacture and in what quantity.
    b) When and from whom to expect deliveries.
    c) To which affiliate to deliver its output.
4) The affiliates execute the orders as instructed, until the final product is delivered to the leader who responds with payment.
5) The leader sends the product to the client, who responds with payment.

**Distributed architecture.** This architecture has no leader to regulate the behavior of the enterprise: All decision making is distributed among the affiliates. This includes decisions to join an enterprise or depart from it, what to manufacture, who to order from, and what prices to charge. Since optimization is done at the time product orders are received, a quoting phase is necessary:

1) A client selects a product from the *E_Public* catalog and sends the quote service a request for proposals.
2) The quote service distributes the request to the affiliates that are listed as suppliers of the product, assigning a deadline to the request.
3) An affiliate who wishes to respond, prepares a proposal. If the product is elementary, the affiliate simply responds with the associated parameters (including price). If the product is composite, it propagates its own requests for proposals:
    a) It examines its *L_Plan* table to locate the components of the product, and submits to the quote

service requests for proposals for each component.
   b) The quotes received by the deadline are sent to the optimizer.
   c) The solution provided by the optimizer is used to prepare the response of the affiliate.
4) These steps repeat until the quote service returns its top solution to the client.

The production phase is similar to the workflow of the autonomous architecture, except that a leader is no longer involved:

1) The client submits an order to the winning affiliate.
2) The affiliate launches a production:
   a) It consults its *L_Plan* table, which has been updated in the quoting phase, and sends the suppliers the appropriate orders.
   b) When these orders have been fulfilled, the affiliate assembles the product and delivers it to the ordering affiliate.
   c) The ordering affiliate acknowledges with payment and proceeds to assemble its own product.
3) These steps repeat until the client receives the final product and responds with payment.

## VII. Example

To illustrate the fundamental differences among the three architectures we present a small example. A number of suppliers and manufacturers in the furniture industry decide to form a collaborative network to fulfill an order for a certain number of chairs by a given deadline. Suppliers $S_1$ and $S_2$ can provide wood legs and backs for the chairs (in different quantities and prices), whereas supplier $S_3$ can provide only leather chair seats. In addition, manufacturers $M_1$ and $M_2$ can assemble chairs from these parts at their set costs; however, each manufacturer is capable of producing only half of the requested number prior to the deadline.

One possibility would be to form a *centralized* collaborative network to be led by $M_1$ (presumably, because it is trusted and experienced). $M_1$ collects production plans from the other affiliates and puts together an optimal plan that produces the chairs with lowest total cost. It orders the necessary chair legs, backs and seats from $S_1$, $S_2$ and $S_3$, arranges their delivery to $M_1$ and $M_2$, and finally delivers the assembled chairs to the end customer.

In another scenario, $M_1$ may still be elected to lead the production, but the other participants are allowed to keep a certain level of autonomy by choosing their own suppliers. For example, $M_1$ may prefer to order chair legs from $S_1$ and chair backs from $S_2$. In such cases the *autonomous* architecture seems to be appropriate. The individual affiliates (manufacturers or suppliers) would need to calculate properties such as prices and risks for the products they offer and advertise them within the network. This information would guide other affiliates in formulating their production plans. The leader is still responsible for communicating with clients and for launching the manufacturing process.

In situations where enterprises prefer an even higher level of autonomy and commitment is limited to individual transactions, a *distributed* architecture would be attractive. In this architecture the manufacturing process is not supervised by a leader. The process begins when a client presents its request for chairs to the entire community, asking for competitive quotes. If a manufacturer decides to participate it propagates its own requests for quotes from other affiliates. Once received, it determines its production plan and submits its proposal to the client. When the client accepts a proposal, it triggers a production phase: An order is sent to the selected manufacturer, and subsequent orders are propagated to other manufacturers and suppliers. Eventually, products are returned in the reverse direction, culminating in a delivery to the client.

## VIII. Challenges

The platform that we described raises several challenging issues that must be addressed to guarantee successful operation. We describe here three such challenges and sketch our approach.

### A. Irregular Behavior

Until now we assumed smooth, fault-free operation. In practice, however, various things could go wrong and the enterprise must respond accordingly. We identify two major types of irregularity: abrupt scale-down and lack of response.

At times, an enterprise must be *scaled down*. For example, in the autonomous or centralized architectures the leader may decide to terminate an affiliate or dismantle the enterprise altogether. In any of the architectures, an affiliate may decide to withdraw a product version or quit the enterprise and dismantle its division. The preferred way for scaling down is to perform these activities *gracefully*; for example, before quitting, an affiliate waits until all its offerings expire, and then satisfies all pending orders. However, at times, scale-down may be *abrupt* rather than graceful. For example, an affiliate who decides to quit instantly would withdraw its product offerings, refuse new orders, and *cancel* both orders that had already been accepted. In cases of abrupt scale-down, production chains are "disconnected" at a particular node. This results in delivery cancellations propagating in the reverse direction. Eventually, the cancellation reaches the affiliate who launched the production. Each manufacturer orders parts from its preferred suppliers, and forwards the finished product to the manufacturer that ordered. The finished chairs are then delivered to the lesser and eventually to the ordering client.

Another type of irregular behavior is *lack of response*. This could happen during *production exchanges* among affiliates or during *management exchanges* between the leader and an affiliate. Examples of the former case are an affiliate who does not acknowledge an order, an affiliate who does not fulfill an order that had been previously accepted, or an affiliate who does not acknowledge a delivery with a payment. In these cases a similar disconnection in the production chain is detected (following a time-out period). This results in a similar

wave of delivery cancellations. Examples of the latter case include not responding to invitations or termination notices.

Workflows are defined to react appropriately in all these occasions.

### B. Risk

The consequence of irregular behavior is that some transactions do not complete as expected. For the benefit of external clients and the participating affiliates, failures should be tracked and advertised. Towards this goal, we formulate two related concepts: *affiliate reliability and risk* and *product reliability and risk*.

**Affiliate reliability and risk.** Each affiliate $A$ is associated with a *reliability* score $reliability(A)$ in the range 0–1. It is the probability that future transactions executed by $A$ will be problem-free. This predictive value is calculated from the affiliate's past performance. It is updated by the performance tracking service at the end of each transaction in which $A$ was involved. Not delivering a product or not delivering it on time, or not acknowledging an order or a delivery, result in a lower reliability score, whereas every prompt execution of a transaction results in a higher score. The complement of this score is the risk associated with this affiliate: $risk(A) = 1 - reliability(A)$. These scores are posted in the $E\_Marketplace$ table.

**Product reliability and risk.** Affiliate reliability reflects the overall performance of an affiliate. This performance, however, could vary substantially for the different products that it offers. To address this, the performance tracking service also maintains for every product version $P$ that is offered in the *G_Availability* table a product reliability score *reliability(P)* in the range 0–1. It is the probability that future transactions that order $P$ from $A$ will be problem-free. This predictive value is calculated from past orders of $P$ from $A$. Delayed delivery or non-delivery result in a lower reliability score, whereas prompt delivery results in a higher score. As before, risk is the complement of reliability: $risk(P) = 1 - reliability(P)$. Product risk is an important parameter of the product version. It is advertised in the *E_Public* and *G_Availability* tables, along with parameters such as price and time-to-delivery, and it is taken into account during procurement optimizations. For example, the same affiliate could offer different versions of the same product, where the version that involves higher risk is offered at a lower price.

If a complete production is diagrammed as a graph in which each participating affiliate is a node and each internal order is an edge between two nodes, then affiliate risk can be viewed as the risk of *node failure* and product risk can be viewed as the risk of edge failure.

### C. In-advance Optimization

The advantage of in-advance optimization (as in the autonomous architecture) is that prices and other product parameters can be disclosed in the product catalog for instant ordering. In contradistinction, on-demand optimization either uses a quoting phase that requires a delay for retrieving this information (as in the distributed architecture), or the customer simply orders "blindly" with an assurance of receiving optimal results (as in the centralized architecture). The latter is time consuming as well, because the leader optimizes each order upon receipt.

The downside of in-advance optimization is the possibility that when orders are received, promised prices (and other parameters) may no longer be available due to events that have taken place since the most recent optimization; for example, changes in prices or in the available quantities, or updates to affiliate or product risks.

One solution to this issue is to react to *enterprise state changes* (e.g., an affiliate changes the price of a product that another affiliate plans to use, or the quantity of a product available from a supplier is reduced due to another order) with re-optimization of all the products that are affected by the change. Note that because products may be affected transitively, re-optimizations are likely to cascade throughout the offerings listed in *G_Availability*. To avoid the high cost of frequent re-optimization, it is possible to re-optimize *periodically* (e.g., once a day), or when the level of change exceeds a *threshold* (e.g., when price changes exceed 2%). In such cases the plans that were pre-selected as optimal may be only near-optimal when executed; yet this approach is quite common in many other domains [25], [26]. Note that when available quantities have changed since the last optimization, it is possible that a transaction based on in-advance optimization will fail due to insufficient quantities. Yet, this could also be the case in the centralized architecture.

## IX. Conclusion

Collaborative manufacturing networks can take on different styles, from a tight centralized organization to a distributed discretionary organization. In this paper we isolated four important feautures that affect the style of such organizations, and we combined them in three diverse architectures: centralized, distributed and autonomous. Realizing that no one architecture is suitable for every need, we described a multi-faceted platform that supports the creation and operation of virtual enterprises in any of the three architectures. The platform is designed around three repositories: An information repository to mange the information of the enterprise, a services repository to provide the necessary functionalities, and a workflow repository to support multiple ways of weaving services and information into operational procedures. We also provided additional details on three significant challenges.

The work is ongoing, and its present focus is on the implementation of a platform that meets the challenges we described. We plan to use the Business Process Execution Language (BPEL) for workflow implementation [27]. BPEL is an XML-based workflow definition language for describing business processes that are connected via web services. It decouples the business process definitions and service implementations, thus allowing services replacement and modification without affecting the business process, and encouraging fast

development of various business processes by reusing existing services.

Future research directions that this work could follow include

1) Encapsulating an entire virtual enterprise in an affiliate; that is, embedding virtual enterprises in other virtual enterprises.
2) When and how to consolidate different virtual enterprises into one, and when and how to split them.
3) How to incorporate affiliates that perform functions other than manufacturing; for example, transportation of products.

## REFERENCES


[1] A. D'Atri and A. Motro, "Virtue: a formal model of virtual enterprises for information markets," *Journal of Intelligent Information Systems*, vol. 30, no. 1, pp. 33–53, 2008.

[2] L. M. Camarinha-Matos and H. Afsarmanesh, "A comprehensive modeling framework for collaborative networked organizations," *Journal of Intelligent Manufacturing*, vol. 18, pp. 529–542, 2007.

[3] J. A. Zachman, "A framework for information systems architecture," *IBM Systems Journal*, vol. 26, pp. 276–292, 1987.

[4] The Open Group, *TOGAF Version 9.1*. Zaltbommel, Netherland: Van Haren Publishing, 2011, ISBN-10 9087536798.

[5] A. Martel, J. Geunes, and P. Pardalos, "The design of production-distribution networks: A mathematical programming approach," *Supply Chain Optimization*, vol. 98, pp. 265–306, 2005.

[6] B. Arntzen, G. Brown, T. Harrison, and L. Trafton, "Global supply chain management at digital equipment corporation," *Interfaces*, vol. 25, pp. 69–93, 1995.

[7] T. J. Lowe, R. E. Wendell, and G. Hu, "Screening location strategies to reduce ex-change rate risk," *European Journal of Operational Research*, vol. 136, pp. 573–590, 2002.

[8] J. Ridlehoover, "Applying monte carlo simulation and risk analysis to the facility location problem," *The Engineering Economist*, vol. 49, pp. 237–252, 2010.

[9] Z. M. Mohamed, "An integrated production-distribution model for a multi-national company operating under varying exchange rates," *International Journal of Production Economics*, vol. 58, pp. 81–92, 1999.

[10] Y. Guo, A. Brodsky, and A. Motro, "Optive: An interactive platform for the design and analysis of virtual enterprises," in *Proceedings of EI2N 13, 8th International Workshop on Enterprise Integration, Interoperability and Networking*, ser. Lecture Notes in Computer Science, vol. 8186. Graz, Austria: Springer, Heidelberg, 2013, pp. 199–207.

[11] S. Melkote and M. Daskin, "Capacitated facility location/network design problems," *European Journal of Operational Research*, vol. 129, pp. 481–495, 2001.

[12] L. Wu, X. Zhang, and J. Zhang, "Capacitated facility location problem with general setup cost," *Computers and Operations Research*, vol. 33, pp. 1226–1241, 2006.

[13] I. Research, "Cplex optimizer," http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/, online; accessed 23-June-2014.

[14] F. Inc., "Fico xpress optimization suite," http://www.fico.com/en/products/fico-xpress-optimization-suite/, online; accessed 23-June-2014.

[15] L. Mashayekhy and D. Grosu, "A merge-and-split mechanism for dynamic virtual organization formation in grids," in *Proceedings of IPCCC 11, 20th International Conference on Performance Computing and Communications Conference*. Orlando, USA: IEEE Press, New York, 2011, pp. 1–8.

[16] N. Hormazbal, H. L. Cardoso, J. L. Rosa, and E. Oliveira, "An approach for virtual organisations?? dissolution," in *Proceedings of COIN@AAMAS 09, 8th International Workshop on Coordination, Organizations, Institutions and Norms in Agent Systems V*, ser. Lecture Notes in Computer Science, vol. 6069. Budapest, Hungary: Springer, Heidelberg, 2010, pp. 70–85.

[17] F. Kerschbaum, J. Haller, Y. Karabulut, and P. Robinson, "Pathtrust: A trust-based reputation service for virtual organization formation," in *Proceedings of 4th International Conference on Trust Management*, ser. Lecture Notes in Computer Science, vol. 3986. Pisa, Italy: Springer, Verlag, 2006, pp. 193–205.

[18] M. Papazoglou and W. J. Heuvel, "Service oriented architectures: approaches, technologies and research issues," *The VLDB Journal*, vol. 16, pp. 389–415, 2007.

[19] B. Zhou, H. Zhi-Jun, and J. F. Tang, "An adaptive model of virtual enterprise based on dynamic web service composition," in *Proceedings of CIT 05, 5th International Conference on Computer and Information Technology*. Shanghai, China: IEEE Press, New York, 2005, pp. 284–289.

[20] Y. Rezgui, "Role-based service-oriented implementation of a virtual enterprise: a case study in the construction sector," *Computers in Industry*, vol. 58, pp. 74–86, 2007.

[21] Q. Wu, Q. Zhu, and M. Zhou, "A correlation-driven optimal service selection approach for virtual enterprise establishment," *Journal of Intelligent Manufacturing*, 2013.

[22] S. Petersen and M. Divitini, "Using agents to support the selection of virtual enterprise teams," in *Proceedings of AOIS@AAMAS 02, 4th International Bi-Conference Workshop on Agent-Oriented Information Systems*, vol. 59. CEUR-WS.org, 2002.

[23] L. Camarinha-Matos and H. Afsarmanesh, "Virtual enterprise modeling and support infrastructures: Applying multi-agent system approaches," in *Proceedings of Multi-Agent Systems and Applications*, ser. Lecture Notes in Computer Science, vol. 2086. Springer, Heidelberg, 2002, pp. 335–364.

[24] A. Motro and Y. Guo, "The soave platform: A service oriented architecture for virtual enterprises," in *Proceedings of PRO-VE 12, 13th IFIP Working Conference on Virtual Enterprises (Collaborative Networks in the Internet of Services)*, vol. 380. Bournemouth, UK: Springer, Heidelberg, 2012, pp. 216–224.

[25] A. Botea, M. Muller, and J. Schaeffer, "Near-optimal hierarchical path-finding," *Journal of Game Development*, vol. 1, no. 1, pp. 7–28, 2004.

[26] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri, "The impact of caching on search engines," in *Proceedings of ACM SIGIR '07, the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2007, pp. 183–190.

[27] M. Juric, P. Sarang, and B. Mathew, *Business process execution language for web services (2nd Edition)*. Packt, Birmingham, 2006, ISBN 1904811817.