# The SOAVE Platform: A Service-Oriented Architecture for Virtual Enterprises

Amihai Motro* and Yun Guo

Department of Computer Science
George Mason University, Virginia, USA

**Abstract.** The flexibility and reusability afforded by service-oriented architectures seems to be of singular applicability to the management of virtual enterprises. In this paper we describe the architecture of the SOAVE platform: a Service-Oriented Architecture for Virtual Enterprises. The platform provides the members of a networked community with a set of tools with which they can collaborate on the production of complex products. It allows members to perform enterprise management functions, as well as manufacture enterprise products collaboratively by means of peer-to-peer transactions.

**Keywords:** Service-oriented architecture, virtual enterprise, collaborative network, peer-to-peer transactions, production trees.

## 1 Introduction and Background

A virtual enterprise is a coalition of business entities who collaborate on the manufacturing of complex products. The collaboration is often ad hoc, for a specific product only, after which the virtual enterprise may dismantle. The members of a virtual enterprise possess complementary skills and technologies whose combination is deemed necessary for the target product at hand [2].

Web services are distributed, autonomous, platform-independent software components, often limited in their functionalities, but easily available to other applications through standard protocols, thus hiding implementation details from their consumers. Service orientation is an approach to software design that accomplishes more complex functionalities by integrating such services. In recent years, the Service-Oriented Architecture (SOA) has gained considerable popularity, notably for achieving architectural flexibility at low cost [8].

A closer look into virtual enterprises and service orientation reveals that these two approaches complement each other nicely. Service-oriented architectures are particularly attractive for virtual enterprises for several reasons:

* This paper is dedicated to the memory of Alessandro D'Atri, who introduced me to the subject of virtual enterprises.

L.M. Camarinha-Matos, L. Xu, and H. Afsarmanesh (Eds.): PRO-VE 2012, IFIP AICT 380, pp. 216–224, 2012.

- **Reusability.** Virtual enterprises are typically created for specific products, and are dismantled once these products are no longer in demand, a process that potentially incurs high overhead. With a service-oriented architecture, existing services may be reused, allowing new virtual enterprises to be set up at low development cost. Cost is of particular concern to virtual enterprises, as often they involve small or medium size companies, for whom the cost of building and maintaining customized applications could be prohibitive.
- **Flexibility.** A key advantage of virtual enterprises is their *agility*: the capacity to adapt rapidly to changing market circumstances [9]. The loosely coupled nature of services in a SOA allows applications to easily evolve with the changing requirements. New services can be incorporated and old services can be dropped to achieve the desired enterprise model.
- **Interoperability.** By using standard protocols and technologies, service-oriented architectures support interoperability between clients and services. This is particularly valuable for virtual enterprises, because they bring together independent business entities possibly operating on heterogeneous platforms.

In this work, we describe the SOAVE platform: A virtual enterprise architecture built on the principles of a service-oriented architecture. There have been numerous interpretations of the virtual enterprise paradigm, and ours is derived mostly from the VirtuE model [5]. Similarly, from the numerous interpretations of service-orientation, we adopt two fundamental principles: (1) A relatively small number of services is defined; and (2) enterprise work is carried out with a set of *business processes*, where each business process "weaves" basic services into a complex task, with minimal amount of traditional programming.

Perhaps the most salient feature of SOAVE is that it provides a *formal framework* for virtual enterprises architected in accordance with the service orientation paradigm. This framework defines basic concepts, algorithms, transaction protocols, business processes and services to implement collaborative manufacturing of complex products. It formalizes concepts such as product price, product complexity, time-to-delivery, procurement risk, reliability scores, on time *vs*. late delivery, and failure.

**Related Work.** In the past five-six years there has been increased interest in implementing virtual enterprises with service-oriented architectures. Much of the work is concerned with particular industries but of more relevance to our effort here are projects that describe industry-independent architectures, and we discuss here three such projects. A model of virtual enterprises based on service composition is proposed in [10]. However, the authors interpret a virtual enterprise simply as a composition of services, and the focus is on locating and selecting the appropriate services. In contradistinction, we view a virtual enterprise as a network of business entities, and a service is a software component that assists these entities in performing their work. The virtual enterprise architecture described in [3] provides for several layers and a multitude of modules performing a variety of functionalities. But although some components are labeled "services", the architecture does not conform to the service-orientated paradigm in which complex tasks are achieved by

compositions of basic services. The interpretation of both virtual organizations and service orientation assumed in [4] is more in agreement with the current literature. The authors propose a detailed framework for process management in service-oriented virtual organizations. It consists of multiple layers, and is based on common standards and protocols. In contradistinction, our work here is not concerned with massive software infrastructure as much as with the formal analysis of the fundamental concepts of setting up (and scaling down) collaborative groups, constructing complex products by procuring components from peers, and the flow of collaborative manufacturing with its aspects of risk, failure and recovery.

Section 2 reviews the virtual enterprise model: the basic concepts, the supporting information system and the workflow. Section 3 focuses on the architecture: the business processes, the shared services and the peer-to-peer communications. We conclude in Section 4 with a brief discussion of some of the remaining work.

## 2 The SOAVE Model

SOAVE provides a *formal* framework for collaborative manufacturing, in which basic concepts, algorithms, protocols and metrics can be defined and analyzed.

### 2.1 The Basic Elements: Members Making Products

A *marketplace* is a set of networked business entities that are available for participation in virtual enterprises. Any member of the marketplace can launch a new enterprise; the member then becomes the enterprise *catalyst*. The catalyst invites other members to join the enterprise; each member then becomes an *affiliate* and launches a *division*. Each marketplace member is associated with a *reliability* score that denotes its performance. This score is updated after each collaboration.

The catalyst establishes and maintains the set of *products* that the enterprise will manufacture collaboratively. These include both *end* products to be available to outside clients — the essential purpose of the enterprise — as well as interim products to be available only to affiliates to use as components in more complex products. Collaborative manufacturing implies that each product is a root of a *tree* of components: the internal nodes of the tree are *composite* components, and its leaves are *elementary* components. Each node is associated with the affiliate chosen to deliver it, and each edge indicates a procurement transaction. Only the catalyst receives and delivers external orders. Thus, it is associated with the root of every tree.

### 2.2 Product Versions and Their Properties

The same product may be offered by different affiliates, thus providing procurement alternatives. An offering of a product is called a product *version*. While versions are identical in substance, they are distinguished by six properties:

1. *Price*: This is the purchasing price of the version. It is the purchasing price paid to other affiliates for procuring the necessary components, plus a profit markup determined by the affiliate offering this version.
2. *Time*: This is the promised time to delivery (the interval between the time of ordering and the time of delivery). It is the maximal time to delivery of its components, plus time spent locally to manufacture the product.
3. *Risk*: This is the risk that the product will not be delivered as promised. It combines the risk associated with the procurement of its components and the reliability of the offering affiliate. The precise calculation of risk is elaborated later, and for now we denote it as a function θ of the underlying risks.
4. *Expiration*: This is the time when this version will expire. It is not later than the minimal (soonest) expiration time of the components.
5. *Depth*: This measures the length of the longest path (procurement chain) in the manufacturing tree of this version.
6. *Complexity*: This measures the number of nodes in the manufacturing tree.

Formally, consider a product version $P$ with components $P_1, \ldots, P_n$ which is manufactured by affiliate $A$. Then

$$Price(P) \geq \sum_{i=1}^{n} Price(P_i) \quad (1)$$

$$Time(P) \geq \max_{1 \leq i \leq n} Time(P_i) \quad (2)$$

$$Risk(P) = \theta\,(Risk(P_1), \ldots, Risk(P_n), Risk(A)) \quad (3)$$

$$Expiration(P) \leq \min_{1 \leq i \leq n} Expiration(P_i) \quad (4)$$

$$Depth(P) = \max_{1 \leq i \leq n} Depth(P_i) + 1 \quad (5)$$

$$Complexity(P) = \sum_{i=1}^{n} Complexity(Pi) + 1 \quad (6)$$

Under this scheme, an enterprise may offer a product version that requires a long time and carries a high risk, but has a low price; another version that requires a short time and carries a low risk, but has a high price; and so on. Note that an affiliate can take advantage of new sourcing opportunities available to it, by offering a new version, and on expiration withdraw the older version. As can be seen in inequalities (1), (2) and (4), an affiliate may set *Price* higher than the cost of procurement, to include profit; it may set *Time* higher than the maximal procurement time, to include local manufacturing; and it may set *Expiration* sooner than the lowest expiration.

**Risk Calculation.** *Risk*($P$) is defined as the probability that the affiliate $A$ would not deliver the product $P$ as promised. This could happen either because $A$ did not receive any of its components $P_i$ as planned, or because $A$ itself failed. Hence, it combines *Risk*($P_i$) and *Risk*($A$) (the latter is the complement of the reliability score of $A$). In the product tree, it is convenient to view affiliate failure as *node failure* and procurement failure as *edge failure*. In general, it cannot be assumed that the $n + 1$ components of *Risk*($P$) are mutually exclusive (i.e., it may not be assumed that there is at most one failure) and, *Risk*($P$) must be calculated according to De Moivre's inclusion-exclusion principle [7]. In practice, however, unless $n$ is small, it is impossible to calculate

*Risk*(*P*) in this way, and one must settle for lower and upper bounds, such as those suggested by the Bonferroni inequalities [1]. If we assume that the $n + 1$ events are *independent* (i.e., the failure of a node and the failure of each edge are unrelated), then, using a simplified inclusion-exclusion formula [6], *Risk*(*P*) may be calculated from the risks (i.e., reliability scores) of the affiliates associated with the production of *P*. Confirming intuition, *Risk*(*P*) increases with the complexity and depth of *P*.

### 2.3 The Enterprise Information System

The virtual enterprise information system is based on nine database tables, arranged in three tiers. Tables 1 and 2 are *external*: they are the only tables available outside the enterprise. Tables 3, 4, 5 and 6 are *global*: they relate to the entire enterprise (they are mostly managed by the catalyst) and they are available to all the affiliates of the enterprise. Tables 7, 8 and 9 are *local*: they relate to individual divisions (each division stores and manages a "horizontal slice"). Database keys are underlined. The relationships among these tables are monitored by various foreign key constraints (not shown). Many of the attributes have already been discussed.

1. *Marketplace* (*Member*, *Reliability*, *Description*): The community of potential members available for participation in virtual enterprises. *Member* identifies the member and *Description* provides information on its manufacturing capabilities.
2. *Public* (*Product*, *Version*, *Price*, *Risk*, *Time*, *Expiration*): The public product catalog for placing external orders. *Product* and *Version* identify the product version. (*Public* is a view of *Availability* that shows only end products.)
3. *Catalog* (*Product*, *Description*): The products of the enterprise. *Product* identifies the product and *Description* is a textual description of the product.
4. *Directory* (*Affiliate*, *Reliability*, *Description*): The enterprise affiliates (including the enterprise catalyst). *Directory* is a subset of *Marketplace*.
5. *Availability* (*Product*, *Version*, *Affiliate*, *Price*, *Risk*, *Time*, *Depth*, *Complexity*, *Expiration*): The product versions presently available throughout the enterprise. *Affiliate* is the (unique) manufacturer of the product version.
6. *Orders* (*Order*, *Product*, *Version*, *Affiliate*, *Price*, *Risk*, *Time*, *Depth*, *Complexity*, *Expiration*, *Rtime*, *Dtime*, Status): A log of the orders received by the enterprise. *Order* is a unique identifier. *Rtime* and *Dtime* are the times the order was received and delivered. *Status* is "in progress", "completed" or "failed". The other attributes are the values published in *Availability* at the time of the order.
7. *L_Availability* (*Product*, *Version*, *Price*, *Risk*, *Time*, *Depth*, *Complexity*, *Expiration*): A view of *Availability* with versions from a particular affiliate only.
8. *Plan* (*Product*, *Version*, *CProduct*, *CVersion*): For each version offered by this affiliate, the component products it requires and the versions to be procured.
9. *L_Orders* (*Order*, *Product*, *Version*, *Price*, *Risk*, *Time*, *Depth*, *Complexity*, *Expiration*, *Rtime*, *Dtime*, *Status*): A log of the orders received by this affiliate. The attributes are similar to those of *Orders*.

### 2.4 Regular Workflow and Irregular behavior

Collaborative manufacturing begins when a client consults the *Public* table for the available product versions and their essential parameters (*Price*, *Risk*, *Time*, and *Expiration*) and sends an *Order* message to the catalyst for a particular product version. After verifying the validity of the order, the catalyst acknowledges it with an order number and sends an *Order* message to the manufacturing affiliate. The affiliate launches a production: It consults the product's *Plan*, which describes the components necessary and their chosen providers, and sends the providers *Order* messages. When all orders have been fulfilled, the affiliate assembles the product and sends a *Delivery* message to the ordering affiliate (presumably with an invoice and shipment tracking information). The ordering affiliate acknowledges (with payment information) and proceeds to assemble its own product. This continues until the catalyst receives the finished product, which it sends to the client, who responds with payment.

This workflow assumed smooth, fault-free operation. In practice, however, various things could go wrong. We identify three basic types of *irregular behavior*. The business processes are defined to manage these behaviors appropriately.

At times, an enterprise must be *scaled down*: The catalyst may want to withdraw a product, terminate an affiliate, or dismantle the enterprise altogether; similarly, an affiliate may want to withdraw a product version, or 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 may decide to quit instantly, withdraw products before expiration, refuse new orders, and cancel orders that were accepted from others or issued to others. In such cases production trees are “disconnected” at a particular node with delivery cancellations propagating up the tree all the way to the root. Another type of irregular behavior is *lack of response* during *production exchanges* among affiliates. An affiliate does not acknowledge an order, does not fulfill an order that has been acknowledged, or does not acknowledge a delivery (with payment). In these cases a similar disconnection in the production tree is detected (after a time-out period). This results in a similar wave of delivery cancellations. A third type of irregular behavior is *lack of response* during *management exchanges* between catalyst and affiliate; for example, not responding to invitation or termination notices.

## 3 The SOAVE Architecture

We describe a particular platform, yet it is important to note that the architecture allows deviation from this configuration, by customizing the business processes and the services that they deploy. SOAVE provides each marketplace member with the necessary tools to (1) launch a new enterprise as a catalyst or join other enterprises as an affiliate, (2) perform management functions, and (3) perform day-to-day operations (collaborative manufacturing of products). All members have identical configuration (clones), allowing them to function as catalysts or affiliates in different

enterprises. The catalyst does not manufacture — it only supervises and leads. Possibly, a regular affiliate could be co-located with the catalyst.

The platform is based on three concepts: *business processes*, *services*, and *messages*. Each member has access to an identical set of business processes that perform management functions and day-to-day operations. These processes involve limited "internal logic" and most work is performed by a collection of predefined services, available from a single *service repository*. Members are able to communicate with each other as necessary, using a fixed set of message types.

### 3.1 Business Processes

Presently, SOAVE defines 13 processes for either management or production.

**Management Processes.** There are six global-level processes for catalysts and five local-level processes for affiliates. The global processes are: launch a new enterprise, dismantle an existing enterprise, invite a new affiliate, terminate a current affiliate, offer a new product, and withdraw a current product. The local processes are: launch a division, dismantle a division (quit), offer a new product version, withdraw an expired product version, and renew an expired product version.

**Production Processes.** There are only two production processes: (1) External order processing is executed by catalysts; it describes the process that begins with an *Order* message received by the catalyst from an external client, and ends with the catalyst delivering the product to the client. (2) Internal order processing is executed by affiliates; it describes the process that begins with an affiliate receiving an *Order* message from another affiliate, and ends with the affiliate delivering the product.

### 3.2 Services and Messages

The aforementioned business processes require frequent access to the enterprise information service, to look up information and to update it. Of the 13 services presently defined in SOAVE, nine are dedicated to servicing requests for each of the nine database tables. They create or destroy tables, search and retrieve information, and modify their contents. We focus here on the other four services.

**Optimization Service.** This service is called by the business process for offering a new product version. It receives a product version from the affiliate, locates the corresponding plan devised by the affiliate, and finds the best procurement options for the plan components. The optimization parameters are *cost*, *risk*, *time complexity*, *depth* and *expiration*. The service can perform two types of optimization. (1) The affiliate sets limits on all but one parameter, and the service finds the procurement that optimizes the remaining parameter; for example, search for the best price, while not exceeding specific risk or time. (2) The affiliate defines a weighted combination of the parameters, and the service finds the procurement that optimizes it.

**Expiration Service.** This service is called when an enterprise is launched. It monitors the *Availability* table for expirations. When a product version expires, it

nullifies *Price*, *Risk*, *Time*, *Depth*, *Complexity* and *Expiration*, and notifies the affiliate. When an affiliate receives an expiration notice, it could either withdraw the version (delete the *Availability* row), or re-optimize it (update the row).

**Performance-Tracking Service**. This service is called by the catalyst upon the completion and delivery of each external order. When the values in the relevant row in *Orders* maintain *Dtime* – *Rtime* > *Time*, the order was delivered late. However, it remains to be discovered which affiliate on the production tree *introduced* lateness and which affiliate simply *propagated* lateness. By examining the global *Orders* table and local *L_Orders* and *Plan* tables, the service can assess the performance of each participant and adjust its *Reliability* scores accordingly. Recall that these scores are used in future calculations of risk.

**Failure-Tracking Service.** This service is called by the catalyst upon the receipt of a delivery cancellation (see Section 2.4). By examining the global *Orders* table and local *L_Orders* and *Plan* tables, the service can detect the affiliate responsible for the failure and adjust its *Reliability* score accordingly.

**Messages.** Finally, SOAVE provides templates for five message types. *Invite* and *Terminate* are sent by the catalyst to affiliates, and *Quit* is sent by an affiliate to the catalyst. *Order* and *Delivery* are exchanged between affiliates to launch and complete procurement transactions. The acknowledgement of *Order* includes an order identifier and the acknowledgment of *Delivery* includes payment information.

## 4 Future Work

A pilot implementation of SOAVE has been completed, showing the viability of the overall approach and pointing where the platform could be strengthened. Work is underway to extend the platform in different directions, and we mention here only two important extensions.

**Performance Indicators and Triggers.** One of the most salient features of virtual enterprises is their *agility*: The ability to adapt and transform the enterprise according to market behavior. In SOAVE this is accomplished with the help of performance indicators [5] — statistics that are collected from the information system while the enterprise is operating; for example, the average turnaround time from order placement to fulfillment; the ratio of late deliveries; the affiliates with highest failure rate; the most severe bottlenecks in the production process, and so on. These performance indicators are deployed to trigger new business processes. For example, when an affiliate receives too many orders for a product, a new member would be invited with similar manufacturing capabilities; or when an affiliate misbehaves (for example, has a high ratio of failed transactions), it would be terminated.

**Constitutional Rules.** Constitutional rules [5] are constraints that must be enforced throughout the life of the enterprise. With the use of constitutional rules, virtual enterprises of different “flavors” may be formed; for example, rules can be used to regulate the extent of affiliate autonomy or the degree of competitiveness within an

enterprise. Constitutional rules can be specified as semantic constraints on the database tables (and on the performance indicators that are derived from them). Violations of these rules can be avoided by blocking the violating activity, or they may trigger a compensating business process. For example, assume a rule that a product cannot be offered in too many versions; when a business process attempts to offer a new version, it would be blocked.

## References

1. Bonferroni, C.E.: Teoria statistica delle classi e calcolo delle probabilità. Istituto Superiore di Scienze Economiche e Commerciali di Firenze 8, 1–62 (1936)
2. Camarinha-Matos, L.M., Afsarmanesh, H.: Brief historical perspective for virtual organizations. In: Virtual Organizations—Systems and Practices, pp. 3–10. Springer (2005)
3. Cui, W., Meng, X., Liu, S.: A service-oriented architecture of virtual enterprises for manufacturing industry. In: Proc. CSCWD 2010, 14th Int'l Conf. on Computer Supported Cooperative Work in Design, pp. 373–377 (2010)
4. Danesh, M.H., Raahemi, B., Kamali, M.A.: A framework for process management in service-oriented virtual organizations. In: Proc. NWeSP 2011, 7th International Conf. on Next Generation Web Services Practices, pp. 12–17 (2011)
5. D'Atri, A., Motro, A.: VirtuE: a formal model of virtual enterprises for information markets. J. Intelligent Information Systems 30(1), 33–53 (2008)
6. D'Atri, A., Motro, A.: Virtual enterprise transactions: a cost model. In: Information Systems: People, Organizations, Institutions, and Technologies, pp. 165–174. Physica-Verlag (2010)
7. De Moivre, A.: Doctrine of chances – a method for calculating the probabilities of events in plays. Pearson, London (1718)
8. Erl, A.: Service-oriented architecture: concepts, technology, and design. Prentice-Hall (2005)
9. Goranson, H.T.: The agile virtual enterprise: cases, metrics, tools. Praeger (1999)
10. Zhou, B., Tang, J., He, Z.: An adaptive model of virtual enterprise based on dynamic web service composition. In: Proc. of CIT 2005, 5th Int'l Conf. on Computer and Information Technology, pp. 284–289 (2005)