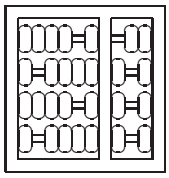

INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Using Choreography to Support Collaboration in Agricultural Supply Chains

E. Bacarin *E. R. M. Madeira*
C. B. Medeiros

Technical Report - IC-07-007 - Relatório Técnico

March - 2007 - Março

Using Choreography to Support Collaboration in Agricultural Supply Chains

Evandro Bacarin* Edmundo R.M. Madeira† Claudia Bauzer Medeiros‡

March 15, 2007

Abstract

This paper presents an approach to support choreography in agricultural supply chains. It depicts a model for this kind of chain that considers both static and dynamic aspects, and their mapping to an underlying architecture. In particular, the model emphasizes mutual agreements, coordination of activities, quality enforcement and activity documentation. The architecture is centered on mapping chain elements to Web Services and their dynamics to the choreography of services. A case study, for soy supply chains, is used to motivate the approach.
keywords: Agricultural supply chains, service choreography, web services, business process collaboration.

1 Introduction

A supply chain is a network of retailers, distributors, transporters, storage facilities and suppliers that participate in the sale, delivery and production of a particular product [7, 8]. It is composed of distributed, heterogeneous and autonomous elements, whose relationships are dynamic. Supply chains present several research challenges, such as recording and tracking B2B and e-commerce transactions, designing appropriate negotiation protocols, providing cooperative work environments among enterprises, or coordinating loosely coupled business processes [1].

As in any kind of distributed process, collaboration within a chain may happen in form of orchestration or choreography. Orchestration defines an interaction flow among different chain elements in a determined business process [10]. Usually, there is a central manager that controls and synchronizes all activities that occur inside the supply chain. Choreography is more collaborative; each participant knows its role in the collaboration scenario and acts accordingly. There is not a manager that controls the collaboration globally.

In most cases, a combination of both is required. There are many new domains in which such coordination issues are treated. In particular, we are concerned with the collaboration

*DC/UEL, *bacarin@dc.uel.br*
†IC/UNICAMP, *edmundo@ic.unicamp.br*
‡IC/UNICAMP, *cmbm@ic.unicamp.br*

processes within an agricultural chains and their mappings to Web service execution. We have already treated the issue of orchestration in a previous paper [3]. This report is devoted to choreography issues.

Agricultural supply chains are a specific kind of chain that has a large economic impact all over the world and is highly regulated. Peculiarities in these chains single them out as very good study subjects for all issues concerning collaboration among distributed elements. Indeed, not only are chain components heterogeneous and autonomous, but collaboration among them is crucial for market competitiveness and for fulfillment of constraints.

Agricultural supply chains have interesting characteristics regarding collaboration between partners. They are inherently autonomous and distributed, since they involve distinct enterprises spread all over the world. These enterprises are embedded in quite different organizational cultures, developing in diverse commercial segments ranging from primary production (e.g. commodities) to finely manufactured items. Agricultural supply chains also demand intense collaboration between partners in order to produce better, cheaper and healthier items.

The main contributions are the following: (a) an architecture for implementing a model that allows collaboration in supply chains through choreography; and (b) the specification of the choreography using Web services, defining the appropriate interfaces and interactions among chain components.

The rest of this paper is organized as follows. Section 2 briefly describes our model and a motivating example used throughout the paper. Section 3 describes the architecture, putting focus on the model elements that allow collaboration – the managers. Section 4 discusses choreography implementation in our architecture. Section 5 shows how the architecture can be used in an agricultural supply chain, given a scenario, the architectural arrangements to implement the scenario, and depicts its execution in terms of choreographed message exchanges. Section 6 presents some related work and Section 7 concludes the paper.

2 A Model for Agricultural Supply Chains

This section presents the main concepts used in the paper and points out some key issues concerning cooperative processes within an agricultural chain. In order to motivate our model and justify our approach, it also presents a simple agriculture supply chain that will be used throughout the paper.

2.1 Motivating example

Figure 1 shows our example – the *soybean supply chain*. The goal of this chain is to process soy and commercialize its products – such as refined soybean oil, mayonnaise, bakery products, epoxy, or inks. The chain's starting point is a "Farm" (Producer) that grows soy. The harvested crop undergoes a so called "rolling" process that produces crude soybean oil and soy flakes as residue. The crude oil, bottled in barrels, is delivered by some sort of transportation means "Transport 1" (T1) to another industry "Mayonnaise Industry" that produces refined oil and mayonnaise. The crude oil can only be processed at this industry if it obeys certain constraints stated in "Regulation 1". The barrels are produced

by the "Barrel Industry". Subsequently, oil and mayonnaise are transported to wholesale and finally retail commercialization, reaching the end consumer. Products and inputs may be stored at different storage facilities throughout the chain – e.g., warehouses. At each stage, various actors – humans or software – may intervene: farmers, lawyers, commodity brokers, health inspectors, quality certifiers or software agents.

Some of the chain's refuse may provide feedback to it, in terms of return flows – e.g., from the Mayonnaise Industry back to the Farm. The Farm may also reuse its own residues, such as, spent flakes to feed cattle.

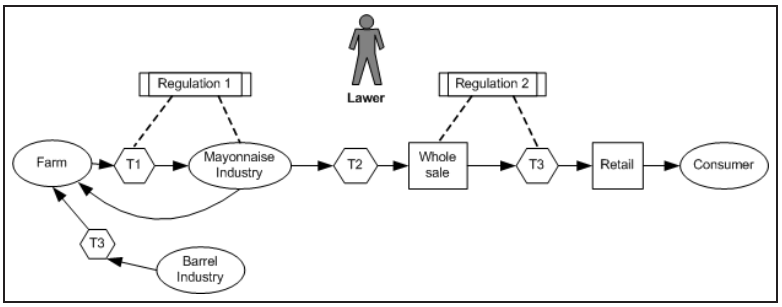

Figure 1: *The Soybean Supply Chain*

Even though the diagram in Fig. 1 shows a sequential execution, this is seldom the case. Each chain component may encapsulate other chains. Collaboration and coordination issues occur at all levels. Coordination may be centralized or distributed among several coordination centers, that negotiate with each other.

2.2 The Model – basic elements

Our model (see [3]) is based on specifying a chain from basic elements, and then progressively constructing their interacting and cooperative processes. The basic elements are Actors, Production, Storage and Transportation. Regulation, Contract, Coordination Plan and Summary are elements needed for providing chain dynamics.

A *Production Element* encapsulates a productive process that uses raw material extracted from its own environment or inputs obtained from other components and transforms such inputs into some product that is passed onwards to the chain. It is represented graphically by an ellipsis.

A *Storage Element* stores products or raw material and a *Transportation Element* moves products and raw material between production and storage components. They are represented by rectangles and diamonds respectively.

Regulations are sets of rules that regulate a product's evolution within the chain. These rules specify constraints, such as government regulations, quality criteria, or conditions determined by a region's social, cultural, economic or even religious context.

Actors are software or human agents that act in the chain. They may be directly or indirectly involved in the execution of activities. A special actor is the *Regulation Certifier*

that is responsible for certifying that activities or products within the chain obey a set of constraints – such as sanitary regulations or quality specifications.

Summaries are elements introduced for traceability and auditability. They are similar to logs, recording chain execution.

Interactions among chain components are organized by means of *Coordination plans* and negotiated via *Contracts*.

Contracts are statements of shared purpose which comprise the mutual obligations and authorizations that reflect the agreements between trading partners [14] that define quality, delivery schedule and costs. They delineate patterns of interaction among the partners.

A *Coordination plan* is a set of directives that describes a plan to execute the chain. This plan may be fully specified before chain execution or, more often than not, be constructed and modified during chain execution. Plans indicate, among others, sequences of chain elements to be activated, and actors responsible for monitoring these sequences. They trigger activity execution, synchronize parallel activities and control the overall product flow.

A chain normally has several plans, that are organized and may interact in different ways. One kind of chain coordination is a strict hierarchy of plans where a top level plan activates lower level ones, which must report to the top plan. In the example, for instance, a top level plan would dictate product flow from Farm to Industry via transport. All interactions among these elements are controlled by this top level plan.

Alternatively, a chain may be governed by a set of plans that interact collaboratively without any hierarchical structure. In this case, in the example, Farm, Transport and Industry are autonomous collaborating elements without any top level coordination.

Finally, a chain may have some set of hierarchical plans mixed with nonhierarchical ones — e.g., there is a hierarchy for Farm – Transport but the Barrel Industry acts autonomously.

2.3 Element Composition and Encapsulation

Production, Storage and Transportation elements can be simple or complex. Complex elements are those that can be decomposed into other elements. A complex Production element must include other productive processes, while Transportation and Storage elements cannot encapsulate production elements. Regulations may be atomic or complex, containing other regulations within them.

The degree of composition of the elements depends on the level of detail desired. A soybean cooperative may have a number of warehouses and some transportation elements that move beans among its warehouses. The cooperative itself may be composed of a number of farms, each farm with its own warehouses and transportation system. In our example, the Farm might be composed of other production elements, such as: parcels of soybean fields and a crush industry; warehouses for harvested soybean; and transportation means from the warehouses to the crush industry.

3 The Architecture

The architecture supports the model described in Section 2. It is composed of blocks that encapsulate data and/or services. As seen in Sec. 4, these blocks are mapped to Web services.

3.1 Building Blocks

The basic elements of our model are directly mapped to the architecture's blocks Production, Storage, Transport and Actor. Cooperation, collaboration and negotiation within a chain and the documentation of its activities are handled by Manager blocks. Managers may be totally automated or require human Actor intervention. The architecture has managers for: coordination (CM), negotiation (NM), regulations (RM) and summaries (SM), that respectively handle coordination plans, contract settlement, regulations and summaries, all mentioned in Sec. 2.

Distinct kinds of repositories are needed to store information on: chain Participants (the basic elements), Products, Regulations, Contracts and Summaries.

Manager blocks guide the collaboration in the supply chain. Some patterns of interaction are described in Contracts, which are negotiated by Negotiation Managers. Constraints on the supply chain activities are described in Regulations, verified by Regulation Managers. Contracts determine how and when a specific interaction may happen, while regulations may prevent interactions that disobey specific constraints or even force interactions, to enforce other constraints. Contract negotiation and Regulation enforcement are started by Coordination Managers in order to assure an effective collaboration among partners.

3.2 Relationship between managers

As mentioned in Section 2, the interactions among chain elements may be strictly organized in a hierarchy of plans, or be governed by non-hierarchical plans (a flat organization), or a combination of both. Strictly hierarchical situations occur when all interactions are set by a backbone of hierarchically organized Coordination Managers. Such situation is dictated by multiple levels of element composition and determines the so called *scopes*.

In this arrangement, a CM at some level (say, CMa) demands activities to a CM immediately bellow it (say, CMb). CMb has to report back the result of this demand to CMa upon activity termination. Thus, in this scenario, communication between CMs is strictly vertical, in a sense that messages go up or down without skipping any intermediate level. This arrangement is suitable for orchestration.

In the flat organization, all CMs are at the same level. Thus, they can exchange messages among each other freely without obeying any hierarchy. This is suitable for choreography.

In a mixed arrangement, a CM may communicate hierarchically, but also horizontely with CMs at the same level.

Regardless of the kind of organization, the communication among the managers can be classified as synchronous or asynchronous; and peer-to-peer or multicast. Those types of communication follow the usual definitions of distributed systems.

We point out that there is furthermore a need for message multicasting. For instance, a CM may ask other CMs to execute the same activity in order for all of them to reach a common state.

Though the previous discussion focused on CMs, the same kind of communication applies to the other managers. For instance, parallel to a hierarchy of CMs, there is a hierarchy of NMs. Interactions between NMs in this arrangement follows the hierarchy. Also, in case several CMs are at the same level, the correspondingly NMs can interact freely among themselves. Those concepts are illustrated in the following example.

Figure 2 shows the main blocks within the initial segment of the supply chain depicted in Fig. 1. This figure shows that Farm is composed of three other production elements: a parcel of land that grows soy (Parcel), an industry that produces crude oil (Crush), and an industry that bottles the crude oil in barrels (Bottle). Farm also has a Coordination Manager (CM1) that coordinates its activities. The farm negotiates contracts through Negotiation Manager NM1 and also deals with summaries and regulation using its own Summary Manager (SM1) and Regulation Manager (RM1). This arrangement shows our concept of *coordination scope*: CM1 defines a scope, in the sense that it can see and coordinate the elements within its scope. SM1, RM1, and NM1 are inside CM1's scope, meaning that they are prompted by CM1 to start their specific processes. The Mayonnaise Industry has similar arrangements. Note that it is composed of a production element that refines crude oil (Refine) and another that produces mayonnaise from the refined oil (Mayonnaise). Parcel, Crush and Bottle may also have their own Coordination Managers and so on, in a top down manner.

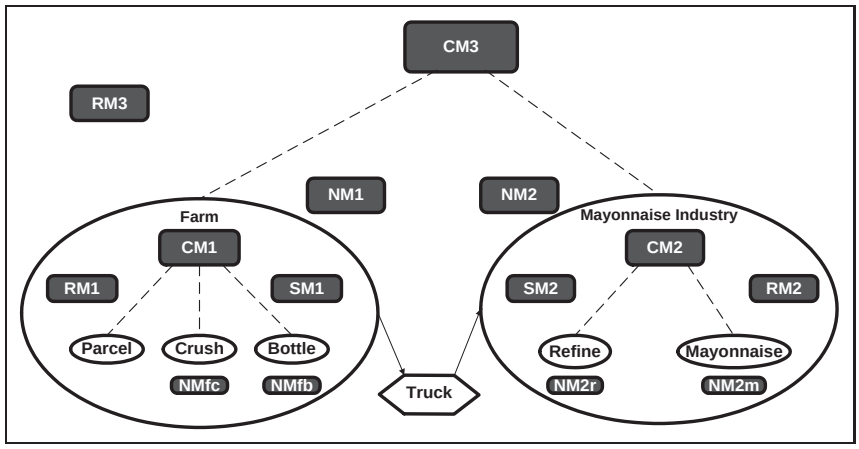

Figure 2: *Managers relationship*

As mentioned previously, strictly hierarchical situations occur when all interactions are set by a backbone of hierarchically organized Coordination Managers. For instance, in Fig. 2, we consider a third Coordination Manager CM3 that controls (orchestrates) all the interactions among the farm, the industry and the transport. Some interactions may cause other interactions among the elements inside the Farm (Parcel, Crush, Bottle). Those interactions are controlled by CM1. Futhermore, Bottle should have other elements within

it and its own Coordination Manager to control their interactions (not shown in the figure), and so on. A similar hierarchy exists for the Mayonnaise Industry.

Hierarchical orchestration is not the only kind of interaction found in an agricultural chain. Given the large number of players in such chains, their scope, temporal and geographical constraints, it is often impossible to rely exclusively on orchestration.

Indeed, Coordination Managers may exchange information and perform cooperative work in a choreography. For instance, CM1 and CM2 may work collaboratively, exchanging messages without intervention from CM3. Orchestration and choreography may happen simultaneously when CM1 and CM2 follow a choreography, while CM1 orchestrates all activities inside the farm.

In another paper ([3]) we detailed orchestration protocols among all the managers, in a rigid hierarchy. Here, we approach what happens in real life: we analyze the choreography among CMs. Each CM, however, still orchestrates the collaboration among all the managers within its scope. Negotiation is always started by some CM that prompts the NM within its scope to start negotiation.

Figure 3 shows an example of this. In this scenario, the Farm negotiates a contract with the Mayonnaise Industry for supplying bottled crude oil under some constraints. The Mayonnaise Industry starts the negotiation process. Its CM (CM2) informs the Farm's CM (CM1) that it wants to start a negotiation (1), and CM1 agrees (2). Both CM1 and CM2 ask their NMs (NM1 and NM2, respectively) to start the negotiation (3 and 4). NM1 and NM2 develop a negotiation process, where NM1 proposes contract clauses to NM2 (5). The latter considers each clause individually and may accept it, reject it or propose an alternative (6). This "proposal X alternative" cycle runs until they agree to or reject the clause, and may ask CM1 or CM2 about negotiation parameters (7 and 8). Eventually, NM1 and NM2 agree on the contract. At the same time, CM1 asks the Crush unit's CM (CMc) to negotiate crude oil supplying with the bottling unit (9). As a consequence of CM1's request, CMc asks NMc (10) to begin negotiation with NMb (11 and 12) to arrange for oil bottling. When the barrel supplying negotiation is finished, NMc reports this to CMc (13), which in turn relays this information to CM1 (14). Note the choreography occurs between CM1 and CM2 (dashed arrows) and the orchestration between CM1 and CMc (bold arrows).

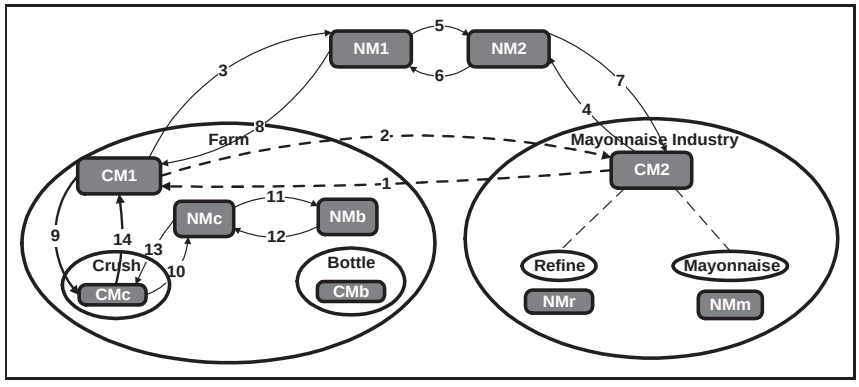

Figure 3: *Choreographed negotiation*

4 Implementation

This section describes issues related to choreography implementation in our architecture. Supply chain components and architectural blocks are implemented as Java classes and are assumed to be published as Web services. The Coordination Manager depends on an underlying engine (e.g., BPEL) to execute a Coordination Plan.

The components of a supply chain (e.g., a Farm, an Industry) provide operations related to their own business (e.g., for selling a box of mayonnaise to a customer). They also implement a set of operations defined by the architecture in order to allow collaboration. Those operations are grouped in interfaces. The classes and interfaces relevant for the scope of this paper are written in Java syntax for readability.

The following sections are organized as follows. Section 4.1 discusses high level issues about choreography among managers. Section 4.2 presents the main interfaces. Section 4.3 considers the Coordination Manager implementation, in special, the Coordination Plan execution focusing choreography. Section 4.4 considers the interation between the Coordination Manager and BPEL.

4.1 Choreography among managers

A component may encapsulate a number of architectural blocks. Component interactions are performed by those blocks. For instance, in Fig. 2, negotiation between Farm and Mayonnaise Industry is performed by NM1 and NM2 that act in lieu of their components.

This paper is concerned with collaboration supported via manager choreography. More specifically, we focus on choreography of Coordination Managers. Choreography, in our approach, happens through messages exchanged among Coordination Managers that are executing a plan, and concerns specific activities in that plan. Messages among managers designate activities, and contain status and parameter information.

A Coordination Manager may execute several instances of the same Coordination Plan. Thus, each plan instance involved in a particular choreography must be uniquely identified. A specific operation called *prepareChoreography* (Sec. 4.2.2) is used to assign this identifier to each instantiation of a plan.

4.2 Main interfaces

4.2.1 Component Interfaces

Interactions among chain components can occur in two ways: directly between two components; or indirectly, mediated by managers. Thus, every chain component must provide interfaces to grant access to its managers, and to allow its managers to contact it back.

4.2.2 Coordination Manager Interfaces

Orchestration and choreography occur by interactions of two or more Coordination Managers. In both cases, one CM (e.g., CM1) executing a coordination plan (P1) asks another

CM (CM2) to execute a new plan (P2). The interface *ChoreographicIF* (Fig. 4) is responsible for interaction among CMs within a choreography, while the *CoordinationIF* (Fig. 5) is responsible for interactions within an orchestration.

```
public interface ChoreographicIF{

  public PlanInstanceIdenfication prepareChoreography(
    CoordinationManagerAddress caller,
    PlanInstanceIdentification callerPlanInstId,
    CoordinationPlanAddress planAddr,
    PlanIdentification planId);
}
```

Figure 4: *ChoreographicIF interface*

Choreographic interaction is started when CM1 calls CM2s procedure *prepareChoreography* of *ChoreographicIF* interface (Fig. 4) and orchestration is started when CM1 invokes CM2s *executeStoredPlan* of *CoordinationIF* interface (Fig. 5). The first procedure does not create a hierarchical relationship between CM1 and CM2, while the second one does: CM1 is the higher level CM and P2 can be seen as a subroutine of P1.

Eventually, CM1 and CM2 may exchange data. In orchestrated interactions, CM1 will receive the result of plan P2 after its completion. In choreographed interactions, CM1 and CM2 will exchange partial results in any point of the plan execution. In order for CM2 to communicate data to CM1, CM2 executes the procedure *reportPlanStatus* of CM1's *ActivityReportIF* interface (Fig. 6).

```
public interface CoordinationIF {

  public void executeStoredPlan(
    CoordinationManagerAddress caller,
    ActivityIdentification activityId,
    PlanIdentification planId,
    CoordinationPlanAddress planAddr,
    Properties pars);
}
```

Figure 5: *CoordinationIF interface*

The Coordination Manager also implements other interfaces, e.g., for administration and connection, not relevant to this report.

In order to execute *prepareChoreography* (Fig. 4), the caller CM informs its address and the instance identifier of the plan it is executing: *caller* and *callerPlanInstance*, respectively. It also informs the plan that the callee should start: where the plan is stored (*planAddr*) and the plan identification (*planId*). Finally, this procedure returns the instance identifier of the newly started plan execution. Now, both caller and callee are executing cooperative plans, each one has the other's instance identification and may exchange messages using *reportPlanStatus*.

The operation *executeStoredPlan* (Fig. 5) has similar arguments. The main difference

is that *activityId* parameter informs the specific activity within the caller plan that will receive the execution status of the called plan.

The operation *reportPlanStatus* (Fig. 6) is used to exchange data between executing plans. This operation conveys a status report message to the target plan (parameter *st*). Such message may contain additional data and is correlated to a specific activity of the target plan (*activityId*). This is exemplified in Sec. 5. The parameter *st* informs a status (DONE, ACTIVE, SUSPENDED, RESUMED, CANCELED). Note that *ActivityIdentification* includes *PlanInstanceIdentification*.

```
public interface ActivityReportIF {
  public void reportPlanStatus(
     ActivityIdentification id,
     PlanStatus st);
}
```

Figure 6: *ActivityReportIF interface*

ActivityReportIF interface also receives reports from other kind of activities in a similar way.

4.3 The Coordination Plan

A coordination plan specifies how a part of a chain is to be executed, being composed of a set of activities. It is specified as an XML file that can be for instance mapped to a BPEL script. The values transferred to and from activities are also XML files. Plan specification requires the following:

- activity specification, i.e., sections that provide details about each activity in the plan;
- synchronization primitives that allow collaboration among activities of different plans.

A plan is executed by instantiation of its XML file using parameters provided by the running environment for this execution. Each instantiation therefore is assigned a unique identifier.

The execution of an activity (A1) in a plan P1 may start the execution of another coordination plan (P2) by a possibly distinct CM (CM2). This new plan P2 receives several parameters to allow its execution, in particular: the address of the CM (CM1) running the caller plan (P1), the instance identification of plan P1 and the identifier of the activity that started plan P2. Using this information, P2 may send information back to P1 with the "sendresponse" primitive. Activity A1 may be synchronous or asynchronous. In the first form, activity A1 blocks until P2 sends a response directed to A1. In the asynchronous form, P1 will not block, but eventually will want to know the status of the action started by activity A1. To do that, P1 executes the "waitresponse" primitive that will block until P2 sends information back to P1.

In a choreographic interaction, at any time P1 sends information to P2 and vice-versa. Thus, both P1 and P2 must know the address and the identifiers of the partner. They

exchange this information using the "chop" primitive at the beginning of each plan. An example of such interactions is shown is Sec. 5. All data exchanged between P1 and P2 is, in fact, intermediated by CM1 and CM2 that control the execution of their plans.

Each activity has an identification and may yield a result after completion. Activities include: execution of another coordination plan, execution of a clause of a contract, verification of a regulation, execution of a Web service operation, and execution of local operations. Activities may be executed sequentially or in parallel and may be synchronized by specific primitives. The execution of an activity may produce a piece of information. This piece of information is referenced by a variable.

The operations of the main interfaces presented in the previous section are invoked during the execution of the activities of a Coordination Plan, as will be shown in Sec. 5.3.

4.4 Coordination Manager and BPEL interaction

The Coordination Manager is not a coordination engine. It depends on an underlying engine (e.g., BPEL) to execute a Coordination Plan. In order to execute a Plan P, the CM prepares the execution environment, translating P to an engine-compatible format and starting the engine with suitable parameters.

In an orchestration or choreography, as mentioned previously, the participant CMs exchange messages to prepare communication. These messages determine different setup on the underlying engine and on the translation of the Coordination Plan.

The setup process handles important issues related to the collaborative process communication, in particular delivering messages to the correct instance of the cooperative processes. BPEL addresses this routing issue through *correlation sets*. A correlation set is a set of message properties shared by all messages that are related to a specific process instance. For example, in a specific purchase process, the customer sends a purchase order to the seller. This order, identified by a unique property called OrderId, is received by a particular seller process. Further interactions about this purchase order must be handled by the same customer and seller processes. Thus, OrderId is used to identify the involved purchase processes. Before translating a Plan to the underlying engine, the CM creates a unique instance identifier. This identifier is included in the correlation set of the translated script.

This approach of having the CM depending on an underlying engine rather than being an engine itself has a number of advantages:

- Agricultural supply chains are inherently autonomous and heterogeneous. It is not practical either to force all chain's elements to use the same engine or to have distinct versions of the same Coordination Plan for each possible coordination engine. We prefer to define a syntax for Coordination Plans suitable for agricultural supply chains and translate on demand the Plan to a script in the underlying engine format.

- We do not need to implement an engine or to get attached to a specific one. Most of the technologies and standards concerning Web services are still under development. Attaching our architecture prematurely to a specific coordination engine may prevent its extensibility.

- If standards or engines evolve, it is easier to adequate the translation mechanism than to modify all existing Coordination Plans.

5 Utilization

This section shows how the architecture and its implementation can be used to support collaboration in an agricultural supply chain, using our example as basis.

5.1 Scenario

Consider again Figure 1 and the following scenario. The Farm grows soybean, harvests it, and processes it to produce crude oil, bottled in large plastic barrels, sent to the Mayonnaise Industry. The barrels are acquired from the Barrel Industry on demand. So, there are two elements that need to cooperate in order to supply crude oil to the Mayonnaise Industry: Farm and Barrel Industry.

Two weeks before soy harvesting, the farm will inform the barrel industry that it will need barrels in three weeks. Futhermore, it will provide information on barrel delivery – e.g., estimated total number of barrels, and required number of barrels per week. Thus, the barrel industry can arrange for the appropriate logistics to meet this specification, and harvesting and barrel production and delivery will occur in parallel.

5.2 Architectural arrangement

The scenario depicted in the previous section is implemented as follows. The farm and the barrel industry are implemented as web services, FarmWS and BarrelWS, respectively. Each web service has one coordination manager: CMf (FarmWS) and CMb (BarrelWS). CMf has the address *uri:farmCM* and executes the coordination plan Pf, while CMb has the address *uri:barrelCM* and executes the coordination plan Pb. In addition, FarmWS supports operations to initiate harvesting a parcel of land (*StartHarvest*) and to process soybeans (*StartProductProcessing*). BarrelWS implements a service that initiates barrel production (*StartBarrelProduction*).

5.3 Choreography execution

Consider the coordination plans depicted in Fig. 8 and 9. Figure 7 shows the execution diagram of the following scenario, where dark retangles represent blocked activities. Suppose CMf starts executing plan Pf. In the begining, it finds a *chop* operation. It causes CMf to ask CMb to execute *prepareChoreography*, informing Pf's instance identification (1). Pf blocks. CMb starts the plan Pb and informs to CMf the instance identification of Pb (pBid). Pf is unblocked. Pb executes the *chop* operation and receives Pf's instance identification. Note that Pb's instance identification is assigned to the *VarPbInst* variable in Pf (Fig. 8). Similarly, Pf's instance identification is assigned to the *VarPfInst* in plan Pb (Fig. 9). Now, both plans may execute in parallel and exchange messages.

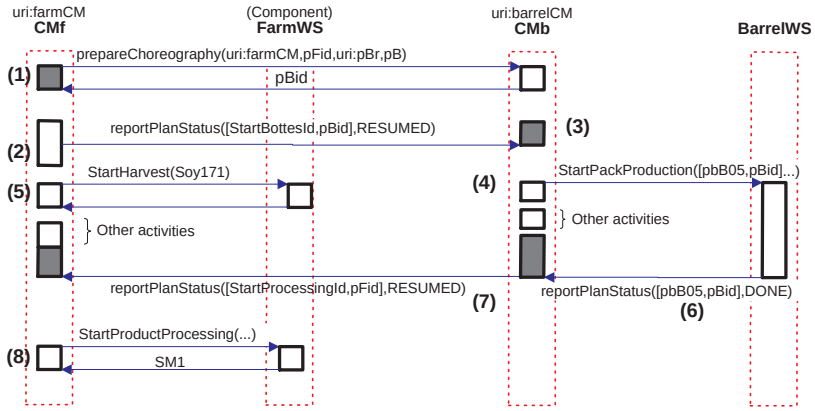

Figure 7: *Choreography scenario*

```
<chop  to="barrelCM" planid="Pb"
   partnerinst="VarPbInst"/>

<!-- Assign BarrelOrder variable -->

<sendresponse activity="StartBarrelsId"
   pi="#VarPbInst" to="barrelCM"
   status="RESUMED"
   value="BarrelOrder"/>

<wsop id="pfA03" name="FarmWS"
   op="StartHarvest">
  <par name="parcel"> Soy171 </par>
</wsop>

<!-- Other activities -->

<waitresponse
   activity="StartProcessing"
   status="RESUMED"/>
<wsop id="pfA09" name="FarmWS"
   op="StartProductProcessing"
   result="SM1">
  <!-- parameters -->
</wsop>
```

Figure 8: *Coordination Plan Pf*

Before start harvesting a parcel of land (Soy171), the farm informs the barrel industry that it will need barrels in a few weeks using the "sendresponse" primitive. This primitive directs a message to Pb's activity *StartBarrelsId*. This message conveys order information (e.g., number of barrels, schedule) contained in variable *BarrelOrder*, the target CM, and the plan instance identification pBid (in variable *VarPbInst*). This operation causes CMf to invoke CMb's *reportPlanStatus* (2). We say that it sends a message *StartBarrelsId* to CMb.

At the same time, CMb reaches the first *waitresponse* primitive (3). This primitive waits

for the message *StartBarrelsId* specified in the *activity* attribute. Note that this attribute matches with the *sendresponse* attribute. If such a message has arrived, Pb collects the order information in variable BarrelOrder (attribute *result*) and continues its execution; otherwise blocks until that message arrives. Next, Pb invokes the Web service operation *StartPackProduction* (4) responsible for managing the production of the ordered barrels.

Meanwhile, the farm performs soybean harvesting invoking FarmWS's *StartHarvest* operation (5) using the *wsop* activity. Attribute *name* is a symbolic name that relates to the Web service location. Such name is declared in a *setup* section (not shown).

```
<chop to="farmCM" planid="Pf"
  partnerinst="VarPfInst" />

<!-- Other activities -->

<waitresponse
  activity="StartBarrelsId"
  status="RESUMED"
  result="BarrelOrder" />
<wsop id="pbB05" name="BarrelWS"
  op="StartPackProduction">
    <!-- parameters -->
</wsop>

<!-- Other activities -->

<waitresponse activity="pbB05"
  status="DONE"/>
<sendresponse activity="StartProcessing"
  pi="#VarPfInst" to="farmCM"
  status="RESUMED" />
```

Figure 9: *Coordination Plan Pb*

In our example, *StartHarvest* is a synchronous operation, but *StartPackProdution* is asynchronous. In this case, synchronization will happen through *waitresponse* primitive. After starting *StartProduction* operation, Pb continues until the next *waitresponse* primitive. When *StartProduction* has ended, it sends back a response message using *reportPlanStatus operation* (6). This message is caught by the *waitresponse* primitive. Note that *waitresponse*'s *activity* atribute matches *wsop*'s *id* attribute.

After sychronization, Pb sends a message (*StartProcessing*) (7) to Pf announcing that the barrels are available. Finally, Pf starts a Web service operation to process the harvested soybean (8).

6 Related Work

Our paper is concerned with (a) supply chain modeling, (b) mechanisms for collaboration among supply chain partners and (c) implementation of our architecture as Web services. We now comment on related work on these issues.

In terms of supply chain modeling, this paper presents a high level model that compre-

hends the main elements of an agricultural supply chain, including quality enforcement and traceability.

Peterson [11] categorizes integrated supply chains into three models, namely: channel master, chain web, and chain organism. The author states that the predominant model in agricultural supply chain is the channel master. In this model, a dominant enterprise specifies the terms of trade across the entire supply chain and the coordinated behaviour is based on specification contracts. The hierarchical nature of our model is well suited to this chain characteristic.

An overview of the USA food supply chain may be found in [13]. It identifies the roles of various members of that supply chain, which are quite similar to the ones depicted in our model. It also mentions that the involvement of US federal agencies in terms of regulation of food production and distribution is increasing, aiming at traceability and prevention of product tampering. Such concerns are also addressed by our model in terms of Regulations and Summaries.

Röder and Tibken [12] propose a method of integrated product and process documentation. Our model has a similar concern; however, while the former work aims at process optimization, ours focuses on quality assurance.

Regarding collaboration mechanisms, inter-organizational interaction among partners raises several issues ranging from high level conversation protocols to implementation particularities.

According to [15], management of inter-organizational service processes is not trivial, because it needs to enable decentralized process execution across heterogeneous workflows, relieving the customer from having to become an expert in the process, and, moreover, customer concerns may change over time. The requirements for the support of inter-organizational processes can be classified in three dimensions: interoperability, flexibility, and customer orientation. Customer orientation is achieved by distinguishing customer-near parts of the service process from background processes. Interoperability and flexibility are achieved via service floats, an XML-based process representation. Service floats exchange process knowledge among service providers. They are sent from service point to service point, and are evaluated at each service point according to a script that defines the activities at the current service point. Although summaries (from our model) and service floats are quite different, they are analogous in a sense that they try to capture some knowledge about the process being developed, specially in terms of the state of the process (pre- and post-conditions), and in terms of documenting the process. Regarding to a summary, the captured knowledge refers to the state and the history of its related product. Regarding to a service float, the captured knowledge refers to the state and history of the activities developed up to the current service point.

Another communication scheme is the framework with an extensible metamodel for specifying different conversations a service supports, presented in [5]. The proposed conversation description model is based on an automata formalism. A conversational act (e.g., Login, CancelPurchase) has associated properties indicating if it is compensable (that is, its effects may be undone after completion), if the act is triggered explicitly by a requester's operation invocation or by a timed transition (e.g., the customer cannot return a purchase after thirty days).

Our model reduces inter-organizational interaction complexity allowing encapsulation of elements. Thus, most interactions are locally concentrated. We also define different kinds of conversational patterns, e.g., for contract negotiation, contract execution, orchestration and choreography.

In terms of implementation, our proposal is based on Web services. Since a business process typically demands a coordinated execution of different services, one aspect has to be considered: composition of these services.

Peltz [10] analyzes issues in service composition and comments on various standards for orchestration and choreography, such as BPEL, WSCI and BPML. Belhajjame et al [4] propose a mechanism for service definition and coordination. Their architecture is based on a 2-level workflow. At the highest level, a workflow orchestrator controls execution, while at the lowest level service execution can be controlled by a regular workflow engine. This is done through entry points placed between activities. In contrast, the work of [2] uses statecharts for defining service composition, and is based on a distributed orchestration engine.

The service-oriented architecture presented in [9] is built upon Web services proposals for inter-enterprise and cross-enterprise integration. Using this architecture, process managers can compose and choreograph business processes based on exposed enterprise and Web services.

Jung et al [6] propose a methodology for business process choreography. Collaboration, in this context, is characterized by a particular *Contract* among the partners. The contract, in this sense, is not an agreement about obligations and authorizations (like ours), but a plan that defines the business logic and message exchanges the participants must perform in the choreography, that is, they are much more similar to our Coordination Plans. The contract's activities are performed by existing workflows described in *Executable processes*. The interactions between Contracts and Executable Processes are specified in an Interface Protocol in terms of five basic interoperability operations that are used by a process to initiate or activate another process' service. They facilitate message exchange or event notifications among business processes.

It is noteworthy that some of those systems mainly focus on orchestration, while others focus choreography. Our work tries to be flexible enough to allow both focus: in a previous paper ([3]) we described an orchestration approach and in this paper we focus on a choreographic approach.

7 Conclusions

This report presented a framework for modeling and coordinating processes in agricultural supply chains. The model takes into account the fact that agricultural chains are inherently heterogeneous, and sensitive to different kinds of constraints. It contemplates the main elements of an agricultural supply chain (producers, transportation, etc) and its dynamics concerning their agreements (through Contracts), the coordination of their activities (Coordination Plans) and quality enforcement throughout the chain (Regulations). The main contributions are an architecture that allows collaboration of supply chain partners

through choreography, and the presentation of how the choreography can be enacted using Web services.

Although a strict orchestration is possible, it is restricted to a small and well-defined segment of supply chains because it requires tightly related partners. Most collaboration is done through choreography since partnership is often occasional and temporary. Thus, an important feature of our architecture is allowing collaboration through orchestration and, in particular, choreography.

Current work includes refining our architecture aiming at traceability issues and at interaction of CMs with different coordination engines.

Acknowledgements

The research reported was partially financed by CAPES, CNPq and Fapesp.

References

[1] A. Arsanjani. Developing and Integrating Enterprise Componentes and Services. *Communications of the ACM*, 45(10):31–34, 2002.

[2] M. Dumas B. Benatallah, Q.Z. Sheng. Environment for web services composition. *IEEE Internet Computing*, pages 40–48, Jan/Feb 2003.

[3] E. Bacarin, C.B. Medeiros, and E.R.M. Madeira. A Collaborative Model for Agricultural Supply Chains. In *CoopIS 2004, LNCS 3290*, pages 319–336, 2004.

[4] K. Belhajjame, G. Vargas-Solar, and C. Collet. Defining and coordinating open-services using workflows. In R. Meersman et al., editor, *CoopIS/DOA/ODBASE 2003*, pages 110–128, 2003.

[5] Boualem Benatallah, Fabio Casati, and Farouk Toumani. Web service conversation modeling: A cornerstone for e-business automation. *IEEE Internet Computing*, 8(1):46–54, 2004.

[6] J. Jung, W. Hur, S. Kang, and H. Kim. Business Process Choreography for B2B Collaboration. *IEEE Internet Computing*, 8(1):37–45, 2004.

[7] K. Kumar. Technology for supporting supply chain management. *Communications of the ACM*, 44(6):58–61, jun 2001.

[8] H. Min and G. Zhou. Supply chain modeling: past, present and future. *Computer & Industrial Engineering*, 43:231–249, July 2002.

[9] S. P. Fremantle, Weerawarana and R. Khalaf. Enterprise Services. *Communications of the ACM*, 45(10):77–82, 2002.

[10] C. Peltz. Web services orchestration: a review of emerging technologies, tools, and standards. Technical report, Hewlett Packard, Co., January 2003.

[11] H.C. Peterson. The "learning" supply chain: Pipeline or pipedream? *American J. Agr. Econ.*, 84(5):1329–1336, 2002.

[12] A. Roder and B. Tibken. A methodology for modeling inter-company supply chains and for evaluating a method of integrated product and process documentation. *European Journal of Operational Research*, 169(3):1010–1029, 2006.

[13] J.R. Stock. *The US Food Supply Chain*, chapter 14. Blackwell Publishing, 2004. In: Food Supply Chain Management, M.A.Bourlakis and P.W.H.Weightman (eds).

[14] H. Weigand and W. Heuvel. Cross-organizational workflow integration using contracts. *Decision Support Systems*, 33(3):247–265, July 2002.

[15] I. Wetzel and R. Klischewski. Serviceflow beyond workflow? IT support for managing inter-organizational service processes. *Inf. Syst.*, 29(2):127–145, 2004.