# Contract e-Negotiation in Agricultural Supply Chains*

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

November 23, 2009

**Abstract**

Supply chains are composed of distributed, heterogeneous and autonomous elements, whose relationships are dynamic. Agricultural supply chains, in particular, have a number of distinguishing features - e.g., they are characterized by strict regulations to ensure safety of food products, and by the need for multi-level traceability. Contracts in such chains need sophisticated specification and management of chain agents – their roles, rights, duties and interaction modes – to ensure auditability. This paper proposes a framework that attacks these problems, which is centered on three main elements to support and manage agent interactions: Contracts, Coordination Plans (a special kind of business process) and Regulations (the business rules). The main contributions are: i) a contract model suitable for agricultural supply chains; ii) a negotiation protocol able to produce such contracts, which allows a wide range of negotiation styles; iii) negotiation implementation via Web services. As a consequence, we maintain independence between business processes and contract negotiation, thereby fostering interoperability among chain processes.

## 1 Introduction

A supply chain is a network of retailers, distributors, transporters, storage facilities and suppliers that participate in the sale, delivery and production

*Research financed by Brazilian Science Foundations CAPES, CNPq and FAPESP.

†bacarin@dc.uel.br, Computer Science Department, University of Londrina, BRAZIL.

‡{edmundo,cmbm}@ic.unicamp.br, Institute of Computing, UNICAMP, 13084-971 Campinas, SP - BRAZIL.

of a particular product [MZ02]. 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 [Ars02].

Trading relations inside a specific supply chain comprise a huge amount of commercial transactions and are subject to legal commitments varying from federal and international laws to particular contracts between trading partners. The use of computational means to perform commercial transactions is increasing steadily. This implies that contracts should ideally be replaced by their electronic counterparts (*e-contracts*), and live negotiation should be performed by software agents (*e-negotiation*). The negotiation process develops in the context of a business process. This raises several interesting problems: i) the efficient execution of a supply chain depends on the commitment of most of its multiple agents – multi-partner negotiation is a must; ii) even though supply chain partners are autonomous and heterogeneous, they must agree on concepts and names; iii) a supply chain demands diverse styles of negotiation – some issues may be resolved through ballots, others through auctions, or through bilateral bargaining, and so forth.

The result of an e-negotiation process is an e-contract to be enacted, in the context of an existing business process. During this phase, partners may want to know if the contract is being enacted properly, that is, whether the terms of the agreement are being satisfied. This raises a number of issues. Not only the contract, but data about its enactment should be stored and retrieved. During the enactment phase, agreements may be changed through a renegotiation process. This requires contract version management, and causes consistency problems among contracts. All of these are open problems pointed out in the literature.

This paper presents an e-negotiation framework that attacks several of these issues, focusing on problems raised by business processes within agricultural supply chains. Such chains have a number of characteristics that distinguish them from other kinds of supply chains – e.g., they must obey strict governmental regulations; they deal with products that are perishable and may influence health conditions; they may be subject to cultural or even religious contexts. The framework is centered on linking contracts and their negotiation to the underlying business processes, rules, and services. The connection between contracts and processes is established in such a way that they can evolve independently, without requiring the update cascades common to this sort of situation. The framework comprises: i) a model for agricultural supply chains; ii) a negotiation protocol suitable for different

styles of e-negotiation; iii) the definition of an e-contract structure; iv) the design and implementation of an e-negotiation framework based on Web services; v) the design and implementation of an enactment infrastructure. The paper's contributions concern issues (ii) – Section 4, (iii) – Section 3 and (iv) – Section. 5.

The following motivating example will be used throughout the paper. Sky Food is a catering company that delivers meals to airlines. It has established a number of contracts with its clients and suppliers, which also have contracts with other parties. The example concerns milk products, and the paper will discuss two of its several contracts. The first is a two-party contract Sky Food has established with a dairy (supplier) that will provide pasteurized milk and other dairy products. The dairy, in turn, is the client of a milk cooperative. However, the milk is not produced by the cooperative, but by its member farms. Thus, the second contract analyzed is a multiparty agreement that the cooperative has established with the farms it represents.

This scenario poses several interesting problems in contract negotiation and enactment. For instance, though the two contracts are independent, their enactment interferes with each other. Moreover, each farm has daily quotas to meet, to enable the cooperative to fulfill its contract with Sky Food. If a farm fails to meet its quota, the others are expected to step up their production (internal renegotiation of the multiparty contract). Additionally, eventual geographical conditions (e.g., a drought) may affect overall milk production requiring renegotiation of both contracts. These and other issues will illustrate the presentation of the framework.

The paper is organized as follows. Section 2 describes our model for agricultural supply chains and an architecture induced by this model. Section 3 describes the organization of our e-contracts and Section 4 describes the e-negotiation process to produce such a contract. Section 5 discusses some implementation issues and a few basic interactions scenarios, such as ballots, auctions, and quota negotiation. Section 6 discusses related work, mainly on contracts and negotiation. Finally, Section 7 concludes the paper.

# 2 The Model and Basic Architecture

Our framework is based on a specific model for supply chains (see [BMM04]). This model specifies a chain from basic elements, and then progressively constructs their interacting and cooperative processes. The basic elements are Actors, Production, Storage and Transportation. Regulations, Contracts, Coordination Plans and Summaries 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.

A *Storage Element* stores products or raw material and a *Transportation Element* moves products and raw material between production, storage components and other transportation components.

*Regulations* are sets of rules that regulate a product's evolution within the chain. These rules specify constraints, such as government regulations, and quality criteria. *Actors* are software or human agents that act in the chain. They may be directly or indirectly involved in the execution of activities. *Summaries* are elements introduced for traceability and auditability. They are similar to database logs, recording chain events.

Interactions among chain components are organized by means of *Coordination plans* and negotiated via *Contracts*. The latter delineate patterns of interaction among the partners, being detailed in the paper.

A *Coordination plan* is a set of directives that describes a plan to execute the chain, coordinating the activities inside an agent or among several chain agents. These activities can be seen as business processes. Plans indicate, among others, sequences of chain elements to be activated, and actors responsible for monitoring these sequences. They trigger activity execution, execute contract clauses, synchronize parallel activities and control the overall product flow. A chain usually has several plans, that are organized and may interact in different ways.

Figure 1 illustrates the main concepts of our model, showing a simplified chain for the motivating example (Section 1). The chain is composed of producers (e.g., Sky Food), means of transportation (e.g., Transp 1), storage facilities (e.g., Airport Storage), and other actors that perform complementary activities (e.g, Lawyer 1). Regulation 1 is a set of rules describing the quality criteria that the milk delivered by the Dairy must comply with in order to be accepted by Sky Food. The figure shows two contracts, one of which (Contract 1), between the Dairy and Sky Food, will be discussed subsequently. Its clauses include parameters such as price, date and regulations to be obeyed (e.g., Regulation 1). The chain has several summaries that provide product, process and service traceability – see [KMBM07]. Plans (e.g., Coord Plan 1) are executed by coordination managers, which send messages to chain elements to perform a set of activities (e.g., asking the Cooperative to produce milk, or Transp 1 to transport milk from the Cooperative to the Dairy).

Production, Storage and Transportation elements can be simple or complex. Complex elements are those that can be decomposed into other ele-

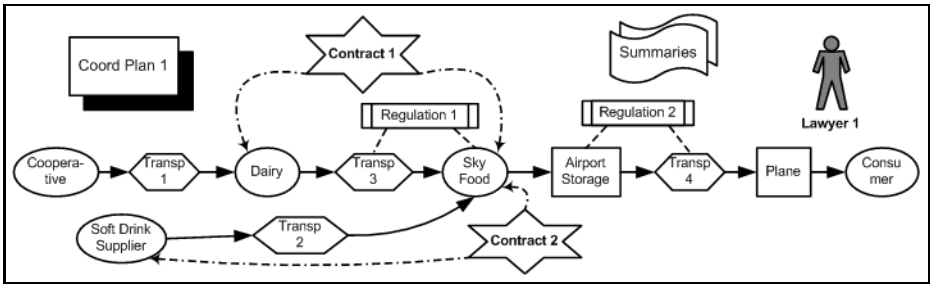


Figure 1: *Simplified supply chain*

ments, e.g., the Cooperative is composed of several member farms (which are Production elements), and a number of storage and transportation facilities. Regulations may be atomic or complex, containing other regulations within them.

The model has been mapped to a Web service architecture, whose components directly reflect the model's elements. 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 previously. Distinct kinds of repositories are needed to store information in: chain Participants (the basic elements), Products, Regulations, Contracts and Summaries. For more on interactions among plans and their composition and encapsulation see [BMM04].

# 3 Contracts

Our e-contract is an instance of a *contract model* and is constructed by a negotiation process. Basically, it is composed of a set of clauses and some auxiliary information that makes it meaningful and supports its execution.

The life cycle of a contract starts with the construction of a contract model, which has a specific purpose (e.g., furnishing of milk by a supplier to Sky Food). Contract models are designed by human actors and stored in specific repositories. They indicate the business rules that must be obeyed when the contract is enacted.

This section presents our proposal for contract specification. Section 3.1 describes our specific model, containing a draft of the contract's clauses with blanks. Whenever two or more partners want to establish a contract, they choose a contract model that better fits their objective. Their respective negotiators (NMs) interact, according to the negotiation protocols proposed

in Section 4 filling the blanks. If the negotiation succeeds, an actual contract instance is produced – Section 3.2. It represents the agreement among the partners and contains the business processes that are to be activated when it is enacted. Thus, a model can originate several instances. Both model and instances are written in XML.

Enactment comprises carrying out the contract's clauses individually and in a proper order. Preconditions are verified before the clause execution. If they hold, a business process associated with the clause is activated. After the business process has ended, postconditions are checked.

Logs are produced during contract enactment, being used to verify contract fulfillment and to help chain traceability. Conditions may change during the enactment. This may demand the renegotiation of the contract. The renegotiation process is quite similar to the primary negotiation, but adds new version control and logging challenges. Discontinued contracts cannot be enacted again, but must be stored with the produced logs for legal purposes. Versioning and log-related issues are outside the scope of the paper.

## 3.1 The Contract Model

Our contract model is a template written in XML. Figure 2 shows a graphical representation of its structure and Figure 3 shows an excerpt of an actual XML document. The graphical representation emphasizes the nesting of the contract's sections and the information needed to construct a final instance: only the information in the gray rounded boxes. Broadly speaking, the contract model has two sections (Figure 2): *property declaration* and *template*. The *property declaration* section lists all the properties that can be negotiated and some specific information about them. The *template* section contains a draft for the contract. The syntax of its content is similar to that of the final contract.

A contract model can be seen as a contract draft with blanks – properties not yet bound to a value. Thus, at its core, the negotiation process consists of agreeing on values (or ranges of values) for each contract property, thereby filling these blanks. At the end of the negotiation, most parts of the contract model (the ones in rounded gray boxes in Figure 2) are used to make up the final contract: mandatory and optional properties are condensed in a single *properties section* in the final contract, the other gray boxes are copied almost verbatim to the final contract.

**The properties.** There are two kind of properties: *contract* and *negotiation*. The former are those used in the final contract. The latter guide the negotiation process itself and are not present in the final contract. A property declaration in the contract model defines what can be negotiated,

whereas in the final contract it determines what can be performed.

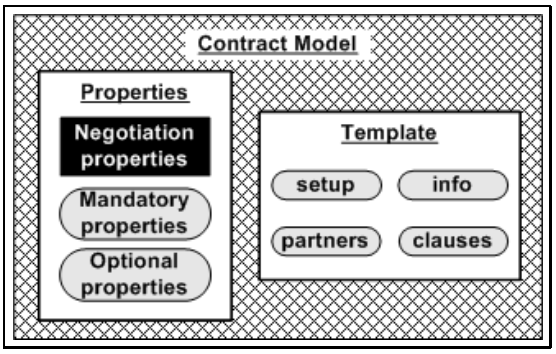


Figure 2: *Contract model schematic representation*

Both kinds of properties may either be negotiated or have a predefined value within the model. Contract properties in the model can be *mandatory* or *optional*. Mandatory properties must be negotiated, in contrast with the optional ones. A negotiation can only be committed after all mandatory properties are negotiated.

```
<cmodel>
  <properties>
    <negotiation>
      <p name="approval-threshold" value="50" type="xs:decimal"/>
      <p name="max-wait-delay" value="?" type="xs:nonNegativeInteger"
          default="5"/>
    </negotiation>
    <mandatory>
      <p name="price" type="xs:float" value="?" dynamics="static"/>
      <p name="amount" type="xs:integer" value="?" dynamics="static"/>
      <p name="deliver-time" type="xs:time" value="?" dynamics="both"
          range="07:00:00,18:00:00" constrained="narrow"/>
      <p name="gis:place" type="xs:string" value="?" dynamics="dynamic"
          enum="maringa,campinas,londrina" constrained="fixed"/>
    </mandatory>
    <optional> </optional>
  </properties>
  <template> </template>
</cmodel>
```

Figure 3: *An Excerpt of a Contract Model, Focusing Properties*

Figure 3 shows an example of properties. Property *values* can be predefined in the model, and thus constant for all contract instances (e.g., *approval-*

*threshold*); alternatively, when assigned to "*?*", they are not bound to a value e.g., *price*). The valid values are defined by the *type* of the property, and restricted by *range* or *enum*.

A value can be bound to the property: a) when the negotiation starts; b) by the negotiation process; c) after the negotiation has ended, when the contract is carried out. In the first case, the value is assigned in the contract model and cannot be changed. The second case is the most common one, in which negotiators agree upon a value for a property. If it is a negotiation property, it will be constant until the end of the negotiation. If it is a contract property, this value will be used in every implementation of that contract. Finally, some property values are assigned only at runtime. This happens for two reasons: a) the negotiators have decided not to establish a fixed value, but agreed on a range of valid values to be assigned at execution time; b) the contract model has established that the value must be assigned only during the implementation of the contract.

A property has two attributes to model all these behaviours: *dynamics* and *constrained*. The *dynamics* attribute has three possible values: "static", "dynamic", "both". The first option means that the property must have a constant value at the end of the negotiation (e.g., *price*). The second (e.g., *gis:place*) means that the value of the property will be defined when the contract is carried out. The negotiators can, at most, define a range of valid values for the property. The last option (e.g., *deliver-time*) means that the negotiators may agree upon a value for the property (static assignment) or postpone the definition to execution time (dynamic assignment).

The *constrained* attribute has two possible values: "fixed" or "narrow". The first (e.g., *gis:place*) forbids any modification on the property constraints (*range* and *enum*). They must be obeyed as they are declared in the contract model. The second (e.g., *deliver-time*) allows the negotiation to narrow the range of the constraints declared in the contract model, that is, the set of valid values is made smaller.

Properties in the negotiation section have standard names and a defined domain of values. For instance, the value "50" for property *approval-threshold* means that, during the negotiation process, an issue submitted to ballot will be approved if it receives more than 50% of the votes. Undefined negotiation properties ("?" value) must have their values negotiated at the very beginning of the negotiation.

If a mandatory or a negotiation property was not negotiated, the *default* value is assigned to it (e.g., property *max-wait-delay*).

**The template.** The template section contains a draft whose syntax is very similar to that of the contract. It has a number of sections that are also present in the final contract. Basically, the content of this section and the

negotiated properties are used to generate the contract instance. Thus, the template section is in the next section.

## 3.2 Contract Instance

A contract is based on a contract model (Section 3.1). The main sections are: *setup*, *info*, *partners*, *properties*, and *clauses*. Excerpts of these sections are shown through examples.

**The setup section.** Contains low level information necessary to the contract implementation, such as: ontologies, paths, libraries, etc. Ontologies play an important role in the negotiation process. They allow the negotiators to understand the meaning of each clause and the relationship among the properties being negotiated. There is a default ontology, and partners can declare other ontologies. In the latter case, terms are preceded by a prefix that identifies the ontology, similar to namespaces within XML documents.

**The info section.** Contains "administrative" information about the contract, such as validity, or the contract model it was originated from.

**The partners section.** A contract establishes rights and obligations among the partners. In general, a binary relationship establishes a duty to a partner and a right to the other partner. However, our contracts allow relationships of higher cardinality.

A partner (Figure 4) can refer to a *person* (e.g., John Doe is the manager of Sky Food) or to a *role* (e.g., the CEO of the Dairy). The identity of a partner is checked against a Participant Repository (Section 2). The address of the repository is a URI. Participants can also be identified by a short name (*abr*).

```
<partners>
  <person name="john-doe" abr="SF" directory="http://www.x.y/pd1"/>
  <role name="ceo-dairy" abr="DRY" directory="http://www.z.k/pd2"/>
</partners>
```

Figure 4: *Contract Partners*

**The properties section.** This section is directly derived from the property declaration in the contract model (Section 3.1). Properties are value containers (like variables or constants). Their values may be fixed (static) or defined during contract enactment (dynamic). Figure 5 presents the property section derived from the contract model of Figure 3. Note that the value of property *price* was agreed to be 0.50, while property *deliver-time* was kept dynamic, but its range was narrowed. Property types are declared in the

contract model. Attributes *dynamics* and *constrained* are only used in the contract model. References to ontologies disambiguate property names (e.g., property *place* belongs to the ontology identified by the prefix *gis*). Type names are also described by ontologies.

```
<properties>
 <p name="price" type="xs:float" value="0.50"/>
 <p name="amount" type="xs:integer" value="200"/>
 <p name="deliver-time" type="xs:time" value="?"
     range="07:00:00,09:00:00"/>
 <p name="gis:place" type="xs:string" value="?"
    enum="maringa,campinas,londrina"/>
</properties>
```

Figure 5: *Examples of Properties*

**The clauses section.** The contract is made from a set of clauses, and is fulfilled when the clauses are executed properly. Each clause (e.g., Figure 6) is identified by a unique identifier within the contract (attribute *id*) and has a number of sections: an *action* name, a *text*, a *dependency* expression, the regulations that must be *enforced* by the execution of the clause, a *service* to be executed, a list of *authorized* and *obliged* partners.

The *action* name describes the task that will be performed when the clause is executed. The *text* dictates rights or obligations to the partners. It contains sentences intending to be both human-readable and machine-processable. Basically, the text contains nouns, verbs and properties. Verbs establish duties among partners and nouns detail them. The properties must be previously declared in the section *properties*. When executed, each embedded property is replaced by the value that was assigned in the property declaration or defined at execution time. Property references embedded in the text are prefixed by the "#" mark. Property references may be used all over the contract wherever applicable. For instance, the *text* section of Figure 6 refers to properties *amount*, *price* and *place*, indirectly, to partners SF and DRY. The word *liter* is defined in the default ontology, whereas *deliver* is defined in the ontology identified by *ecom*. Suppose that there is another clause (id=9) that states that any milk delivered must be paid within a period of three days. The milk referred to in clause 10 is the same as the one in clause 9. This is shown by the notation "milk[9]".

**The depends section** contains a boolean expression that determines if the clause may be applicable. In this example, the condition checks if Sky Food has no debts. If this expression is true, the coordination plan A, stored in a repository whose address is "http://www.coop.com/plans" is allowed to

be executed by the coordination manager at "http://www.coop.com/coop_cm", all of which is informed in the *service* section. The expression may contain arithmetic, relational and boolean operators, as well as external function calls. The coordination plan may demand some parameters. Properties or constants can be assigned to them (e.g., the value of the property *price* is assigned to the parameter *MilkPrice*, Figure 6). Conversely, the section *enforces* is a postcondition that lists the regulations that must be enforced by the execution of the clause.

The enactment of a clause may be started by any of *authorized* or *obliged* partners. An *obliged partner* must accomplish the state of affairs intended by the clause. In the example, the milk must be delivered by the dairy (represented by its CEO). An *authorized partner* has the right to receive the effect intended by the clause. In the example, John Doe (representing Sky Food) has the right of receiving the milk. Both *obliged* and *authorized* items are optional, but at least one of them must be present. Note that if only the *authorized* item is present, it means that the clause conveys an optional right to the listed partners, but none of them is obliged to enact the clause.

```
<clause id="10">
 <action> Milk deliver </action>
 <text>
    @OBLIGED will ecom:deliver #amount liters of milk[9]
    at R$ #price liter to the branch of @AUTHORIZED at #place.
 </text>
 <depends> no_debt(@SF) </depends>
 <enforces> <r> Regulation 1 </r> </enforces>
 <service cmaddress="http://www.coop.com/coop_cm"
       cpaddress="http://www.coop.com/plans/" idplan="A">
  <par name="MilkPrice" value="#price"/>
  <par name="MilkAmount" value="#amount"/>
  <par name="DeliverPlace" value="#place"/>
 </service>
 <authorized partners="@SF" mode="all"/>
 <obliged partners="@DRY" mode="all"/>
</clause>
```

Figure 6: *A Contract Clause*

The example shows a binary relationship: the Dairy must deliver milk to Sky Food. However, our contracts allow relationships with higher cardinality. Note that both *authorized* and *obliged* items may contain a list of several partner names. In this case, the attribute *mode* determines how many of the partners in each list are supposed to perform the task or to be the clause beneficiary. The attribute *mode* has three possible values: "one", "all" or a

positive integer "*n*". The value "one" means that exactly one of the obliged (authorized) partners must fulfill the duty (right). Similarly, "*n*", means that, at least, *n* obliged (authorized) partners must fulfill (should receive) the duty (the right). The value "all" has analogous meaning. The names @AUTHORIZED and @OBLIGED can be used in the *text* item standing for the partners listed in the respective items, according the respective *mode*s. For instance, Table 1 shows two combinations of *mode* assigment.

| **Obliged** | **Authorized** | **Effect** |
|---|---|---|
| one | all | Exactly one of the obliged partners must deliver the established *amount* of milk to each authorized partner. Thus, the obliged partner will deliver *n*amount* liters of milk. |
| all | one | Each obliged partner must deliver *amount* of milk to one of the authorized partners. Some authorized partners may receive *i*amount*, while others may receive no milk. |

Table 1: Examples of *mode* combination for Figure 6

# 4 The e-Negotiation Process

This section describes the negotiation process. It presents a negotiation protocol that can be used for various negotiation styles, e.g., bargaining or auctions. Section 4.1 gives an overview of how negotiations are set up; the subsequent sections detail the negotiation process itself.

## 4.1 Organization of the Negotiation

Negotiators are instances of the Negotiation Managers (NM) of our architecture (Section 2). One of them is the *leader*. A Notary actor is responsible for given bureaucratic chores, e.g., constructing the final contract, or acting as a trusted third-party (e.g., to control ballots).

These players exchange information within a negotiation process through asynchronous messages. The messages may be peer-to-peer or broadcasted. Contract negotiation is directed by the contract model (Sect. 3.1) and has several phases:

1. Negotiation announcement: the Notary announces a new negotiation process or renegotiation of an existing contract. The interested parties register for this process.

2. Leader determination: the leader of the negotiation is chosen. The leader may be predefined in the contract model (the notary just announces it to all negotiators) or chosen by means of an election procedure – in our case, similar to [GM82].

3. Objective announcement: the leader announces the objectives of the negotiation, such as: minimize (or maximize) a property, or enforce a regulation.

4. Negotiation set up: some parameters that guide the negotiation process are set up by means of property negotiation.

5. Restriction announcement: all negotiators may broadcast their restrictions on what is going to be negotiated. For instance, a restriction can be "I accept Price> $10 only if delivery interval < two days".

6. Core negotiation (see Sec 4.2): the contract negotiation takes place. Negotiators exchange messages trying to agree upon property values, through a cycle of proposals and counter-proposals or through ballots.

7. Commit attempt: after the parties have reached an agreement, the Notary verifies if there is any pending issue that prevents the contract to be commited, such as, all mandatory properties must be negotiated. If no problem is found, the negotiation is commited and the final contract can be written down. Otherwise, the negotiation returns to the state just before the commit attempt.

8. Contract (re)construction: after the (re)negotiation is commited, the contract is (re)written by the Notary and is available to be signed by the partners.

## 4.2 Core Negotiation

Core negotiation involves settling values of properties. Contract or negotiation properties may be negotiated by means of two basic mechanisms: request for proposals (RFP) and offers. These basic mechanisms can be combined to provide the various styles of negotiation in a supply chain.

A negotiator A may propose to negotiator B a value (or range of values) for a given property P using an *offer*. B can accept, reject or make a counter-offer. They then engage in a cycle of counter-offers until they agree or give up. Conversely, negotiator A may *request a proposal* from B using an RFP. B answers the RFP sending A an offer that complies with the restrictions of

the RFP, and they may engage in cycles of counter-offers. An RFP or an offer may cover to several properties.

An offer (or an RFP) may be submitted to a *ballot* – see example in Section 5.2.2. The notary broadcasts the offer (RFP) and the list of allowed votes – typically *agree* or *not agree*, for offers, or a list of several options, for RFPs – and waits for the votes. The negotiators send back to the Notary their votes (choices). The Notary counts the votes and broadcasts the result. Negotiators with veto power are listed in the contract model – instead of sending back a vote to the Notary, they may veto the subject. In this case, the ballot is cancelled.

Property values may also be negotiated through an *auction* – see example in Section 5.2.1. In this case, the notary broadcasts an offer or an RFP and collects all answers for them. Then, the leader chooses the answer it considers the best one. Counter-offers are not allowed.

Finally, some negotiators may agree on exchanging information during the negotiation process, bypassing the leader or the Notary – e.g., to establish mutual consensus during the negotiation process. For instance, before answering the RFP, they develop a private negotiation and respond identical offers to the leader. This is useful in case of composition (e.g., the Cooperative and its farms) or for strategic alliance among partners.

## 4.3 Main Protocol Messages

Our negotiation protocol is specified by means of a context-free grammar and state diagrams. The latter describe the steps a negotiator follows after sending or receiving a message.

Figure 7 presents an excerpt of the grammar. Non-terminals are capitalized, terminals are in bold, $\diamond$ means another rule for the same non-terminal, [A] means that A is optional, A+ means one or more A.

The figure shows that the negotiation of a property can be achieved by: (i) making an RFP (line 1), (ii) making an offer (line 2), (iii) issuing a ballot (line 3) or (iv) performing an auction step (line 4). Offers and RFPs are the main negotiation mechanisms. Ballots and auctions use them. These four negotiation primitives may be combined to develop several styles of negotiation.

A negotiator announces an RFP communicating it to one or more negotiators, who will send back a response (line 5). Communicating a message means sending it to a single partner (line 6) or broadcasting it to all negotiators (line 7). The message that communicates the RFP is "new_rfp rfp" (line 8). The terminal symbol "rfp" conveys all information about a specific RFP. This message can optionally disclose the intention of the negotiator, e.g.,

```
1.  NegotiateProperty::= MakeRFP
2.                     ◇ MakeOffer
3.                     ◇ IssueVoting
4.                     ◇ Auction

5.  MakeRFP ::= Communicate RequestForProposal
                ReceiveRfpResponse
6.  Communicate::= send Dest
7.                ◇ broadcast
8.  RequestForProposal::= new_rfp rfp [Obj]
9.  ReceiveRfpResponse::= (RfpResp [MyResp])+
10. RfpResp::= ProposalResponse
11. MyResp::= ProposalResponse
12. RfpId::= rfp_id

13. ProposalResponse::= send Dest proposal agree Offer
14.                   ◇ send Dest proposal no_agree Offer [Reason]
15.                   ◇ send Dest new_offer Offer [Obj]
16.                   ◇ send Dest no_offer RfpId
17.                   ◇ Wait ProposalResponse
18. Wait::= send Dest wait [WaitDuration] [Reason]
```

Figure 7: *Excerpt of grammar for property negotiation*

minimizing the #price property. Offers are made in a similar way (grammar rules were omitted). If the partner agrees to the offer, it sends back a "proposal agree" message (line 13); if it disagrees, it sends back a "proposal no_agree" message (line 14); if it answers an RFP or makes a counter-offer, it sends back a "new_offer" message (line 15); if it does not intend to answer an RFP, it sends back a "no_offer" message (line 16); finally, it may send a wait message informing that it will postpone the answer (lines 17, 18).

Figure 8 shows a possible sequence of messages induced by this excerpt. In this example, the text between triangles represents an RFP and between squares represents an offer. This is not the actual syntax; only the main parameters are shown. RFPs and offers share parameters that allow correlating them. This figure shows that a negotiator (presumably N1) has broadcasted an RFP asking for a proposal for property *price*, considering that property *amount* has value 20. The RFP imposes that a proposal for *price* must be less than 10. Three negotiators answered the RFP, sending back an offer each, with different values for *price*. Figure 9 shows a short bargaining process, where N2 agrees to N1's counter-offer.

Figure 10 shows the steps followed by the leader after it has sent an RFP. The transitions are labeled with the message sent or received. The leader collects offers from other negotiators until all expected offers have arrived or

```
broadcast  new_rfp ◁price?<10; amount=20▷
send n1    new_offer □price=8; amount=20□
send n1    new_offer □price=9; amount=20□
send n1    new_offer □price=7; amount=20□
```

Figure 8: *Example of price survey given an RFP*

```
send n2  new_rfp ◁price?<10; amount=20▷
send n1  new_offer □price=9; amount=20□
send n2  new_offer □price=8; amount=20□
send n1  proposal agree □price=8; amount=20□
```

Figure 9: *Example of bargaining given an RFP*

a fixed length of time elapses. Next, it analyzes all received offers, using the Offer Received diagram (not detailed), and may accept the offer, reject it or make a counter-offer. In this diagram, the prefix "e:" means an internal event, e.g., "e:timeout" means that the leader has not received any offer within a specific time.

Figure 11 shows the negotiator's steps after receiving an RFP. First, it analyzes the RFP and may send back an offer, inform the partner that it will not make any offer, or that it will delay the answer. Prefix "s:" means that the negotiator has sent a peer-to-peer message.

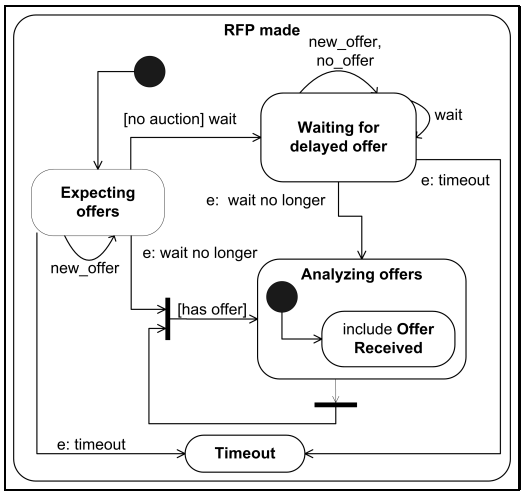


Figure 10: *Leader's state after issuing an RFP*

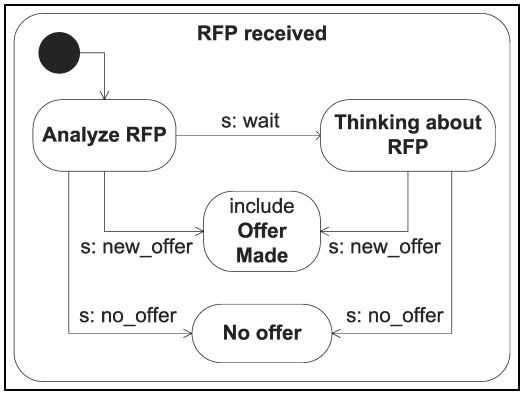


Figure 11: *Negotiator's state after receiving an RFP*

Note that the communication primitives presented in Figure 7 allow sev-

eral styles of negotiation. This flexibility is important in the context of business processes within agricultural supply chains. For instance, a contract model may contain a clause with a property whose value must be agreed on by most of the negotiators (e.g., a maximum production quota for any farm of a cooperative). This is resolved by a ballot. Another clause of the same contract may have a property that must be disputed by the negotiators. This is resolved by auctions, and so forth. Sections 5.2.1, 5.2.2, and 5.2.3 show a few examples of different negotiation styles.

# 5 Some Implementation Issues

Our framework is implemented by Web services, which are suitable for handling heterogeneity, distribution and autonomy of supply chain partners. The messages exchanged among negotiators are mapped directly to Web service SOAP messages. When a message is received by a negotiator, it activates a Web service operation. Our framework specifies a number of operations that enable the reception of the messages needed by the negotiation process. The operations are organized into interfaces.

## 5.1 Interfaces

### 5.1.1 Negotiator Interfaces

Every negotiator has interfaces to receive messages from other negotiators or from the Notary. Interfaces may relate to (1) establishing a negotiation, (2) negotiation setup, or (3) core negotiation.

**Establishment of the negotiation.** The *Advertisement interface* receives messages concerning a new contract negotiation or about the renegotiation of an existing contract. An interested negotiator must register to the negotiation with the Notary. The *Negotiation Coordination interface* receives messages from the Notary concerning the acceptance of that negotiator. This interface follows the WS-Coordination specification [IBM06].

**Negotiation setup.** The *Leader Election interface* receives messages concerning the election of the leader. The *Announcement interface* receives messages from other negotiators announcing their restrictions and objectives for a negotiation.

**Core negotiation.** The *Proposal* and *PeerNegotiation interfaces* receive messages concerning exchange of proposals. They are complementary interfaces by which negotiators may reach a "personal" agreement. The *Voting interface* receives messages concerning a ballot process. The negotiator is asked to vote on some issue and receives the result of the ballot through this interface. Through the *Leader interface*, the leader receives some specific demands.

### 5.1.2 Notary interfaces

Similarly, the Notary has interfaces related to the establishment of the negotiation. Through the *Negotiation Activation interface* the notary is requested (typically by a Coordination Manager) to perform the initial setup of a new negotiation. It follows the WS-Coordination specification. The *Registration interface* is responsible for receiving messages from the negotiators concerning their registration to a negotiation process. This interface also follows the WS-Coordination specification.

The *Leader Election interface* is related to the setup of the negotiation. It receives messages from the negotiator who wants to apply as a candidate in the leader election.

Finally, the Notary helps the core negotiation process by the following interfaces. The *Negotiation* and *Leader interfaces* provide operations needed during the negotiation process, such as, commit or cancel the negotiation, and through the *Ballot interface* the notary is asked to conduct a ballot and to receive the respective votes.

## 5.2 Styles of Negotiation

This section discusses a few of the possible negotiation styles supported. Interactions are enacted by execution of operations of suitable interfaces. The following scenarios show the messages exchanged among the participants of a negotiation and the operations invoked by them. For instance, the first message sent in Figure 12 shows the Sky Food NM sending the message "first answers" to the Notary. This message activates the Notary's "firstAnswersToProposal" operation. Only the main parameters are shown. Acronyms were adopted for brevity's sake. The names of the invoked operations are prefixed with a string that identifies the interface which the operation belongs to. Table 2 shows the meaning of the prefixes. Most of the interfaces are related to the negotiators, a few with the Notary.

| | |
|---|---|
| lif | Leader interface |
| pnif | PeerNegotiation interface |
| pif | Proposal interface |
| vif | Voting interface |
| nbbif | Ballot interface (notary) |
| nlif | Leader interface (notary) |

Table 2: Interface acronyms

### 5.2.1 Auctions

Recall Figure 1. Sky Food wants establish a contract with the supplier that offers the cheapest milk. Two dairies have entered the negotiation: SDA and SDB. Thus, their NMs undergo a negotiation process, led by the Sky Food's NM, directed by a contract model (not shown), and helped by the Notary.

The scenario in Figure 12 depicts a variant of an english auction that aims at minimizing the price (property $p$). The maximum price is 10,00. Thus, (1) the leader asks the Notary to advertise an auction for an RFP and wants the notary to collect at most two answers within 30s. The notary broadcasts the RFP and waits as demanded. The negotiators receive the RFP and (2) send offers to the Notary in response. The notary collects them and (3) sends them to the leader. Now, the leader chooses the best offer (p=8) and (4) asks the notary to start a new auction round. This is repeated until no negotiator answers the RFP of the last round. Finally, (5) the leader agrees with the best offer of the previous round (p=6). This scenario may also be implemented by a Dutch Auction: the leader announces descending offers and the negotiator who first agrees with the current offer wins the auction.

Similarly, the so-called double auction is easy to be developed using these primitives. For instance, the negotiators willing to *sell* a product (in fact, to define a value for a property) send offers to the leader. Conversely, the negotiators that want to *buy* a product (in fact, to ask for a value for a property) send RFPs to the leader. The leader matches RFPs and offers using some criterion and forwards the selected offer to the RFP originator.

### 5.2.2 Voting

Recall again Figure 1. The Cooperative is the negotiation leader and has a number of member farms (F1, F2,...). The Cooperative itself does not produce milk. In order to provide milk to its customers (the Dairy), it negotiates quotas with its member farms. However, the Cooperative wants to avoid that only a few members monopolize the milk market. Thus, it negotiates a maximum quota for any farm before negotiating any contract

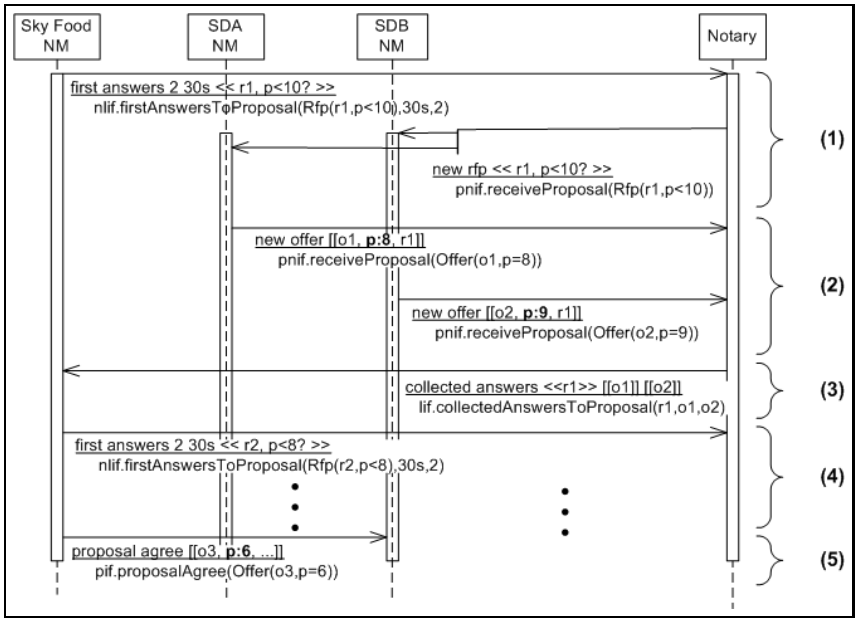


Figure 12: *The english auction*

with its costumers. This is done through a ballot. The Cooperative proposes three alternatives (e.g., 10, 20, or 30). The members vote in their choices.

Figure 13 shows a diagram for this ballot. First, (1) the Cooperative's NM (leader) asks the Notary to conduct the ballot. The Notary (2) accepts the job and broadcasts the issue to be voted to all negotiators, that is, it broadcasts an RFP asking for a quota value and the three choices available (10, 20, or 30). Next, (3) each negotiator sends its vote the Notary. Finally, (4) the Notary counts the votes and broadcasts the ballot result to all negotiators. The option "20" has won with 38 votes.

### 5.2.3 Quota Negotiation

This section resumes the scenario presented in Section 5.2.2, where a maximum quota was negotiated. Now, individual quotas are established, obeying the maximum one. The Cooperative (leader) negotiates milk quotas among several farms, until reaching the desired total. The negotiation process is aware of the parameters:

- $n$: number of suppliers.
- $Q_T$: total amount of milk to be provided.

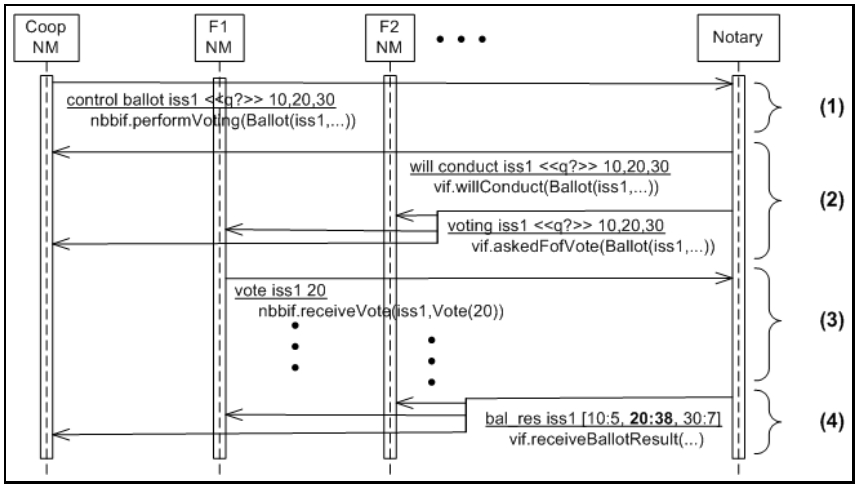


Figure 13: *Voting for maximum quota per farm*

- $Q_i$: amount provided by the $i^{th}$ milk farm.
- $Q_{mi}$: maximum production capacity for the $i^{th}$ milk farm.
- $Q_M$: maximum quota for any farm.
- $P_i$: price (per liter) charged by the $i^{th}$ milk farm to deliver amount $Q_i$

and must obey the restrictions:

- $Q_1 + Q_2 + ... + Q_n = Q_T$
- $Q_i \leq Q_{mi}$ and $Q_i \leq Q_M$

There are two complementary heuristic strategies for quota negotiation: the Cooperative tries to buy cheap milk, while the farm tries to sell expensive milk. The interaction of both strategies usually ends up in an intermediate price. The objective of the negotiation, as determined by the leader, is:

$$minimize \sum(Q_i * P_i).$$

**Leader Strategy**

The cooperative believes that the price asked by any farm depends on the amount of milk. Each farm has a price function that it does not disclose. The strategy of the leader is first to issue a number of RFPs asking the price of different increasing amounts of milk. Based on the answers, for each farm, the dairy constructs, a table $Prices[q, p]$ (Table 3): for a given RFP, the farm

| **T** | **1** | **2** | **3** | **...** |
|---|---|---|---|---|
| **Q** | 10 | 20 | 30 | ... |
| **P** | 0,90 | 0,80 | 0,72 | ... |

Table 3: The price function for a farm

answered that it should deliver $q$ liters of milk at a price $p$ per liter– e.g., the price for milk between 10 and 19 liters is \$0,90 per liter.

Those tables are used to construct the price function for each farm. Next, these functions are used by an optimization tool aiming at finding the minimum cost for the total amount of milk. For the sake of simplicity, we suppose that $Q_i \geq Q_{i+1}$. Let $K$ be number of farms needed to provide the total amount. The equation that describes cost computation is:

$$C_T = Q_1 * P_1 + Q_2 * P_2 + ... + Q_k * P_k \quad (1)$$

The cooperative believes that the prices answered by each farm are inflated. Thus, the cooperative bargains with each negotiator ($i$). It offers increasing prices for the quantity $Q_i$ found for the Equation 1, beginning with a price smaller than $P_i$. The last offer will be the asked price ($P_i$). Supposedly, the farm will agree with one of the offers because the last offer is the one it has proposed.

**Farm Strategy**

The farms follow a different strategy. They listen to RFPs and offers, and respond appropriately. Their strategy is quite simple: when a farm receives an RFP, it returns a higher value. When a farm receives an offer, it accepts offers with values higher (or equal) to the value it expects.

Figure 14 illustrates the interactions between the Cooperative and the farms. The Cooperative (1) broadcasts an RFP inquiring the price each farm charges for 10 liters of milk. Farms F1 and F2 answer the RFP sending offers to the Cooperative. The Cooperative (2) keeps inquiring the farms for different amounts of milk. Next, (3) the Cooperative bargains with F1, and similarly (4) with F2.

# 6 Related Work

This section reviews related work on supply chains (Section 6.1), electronic contracts (Section 6.2), the negotiation process (Section 6.3), and contract enactment (Section 6.4).

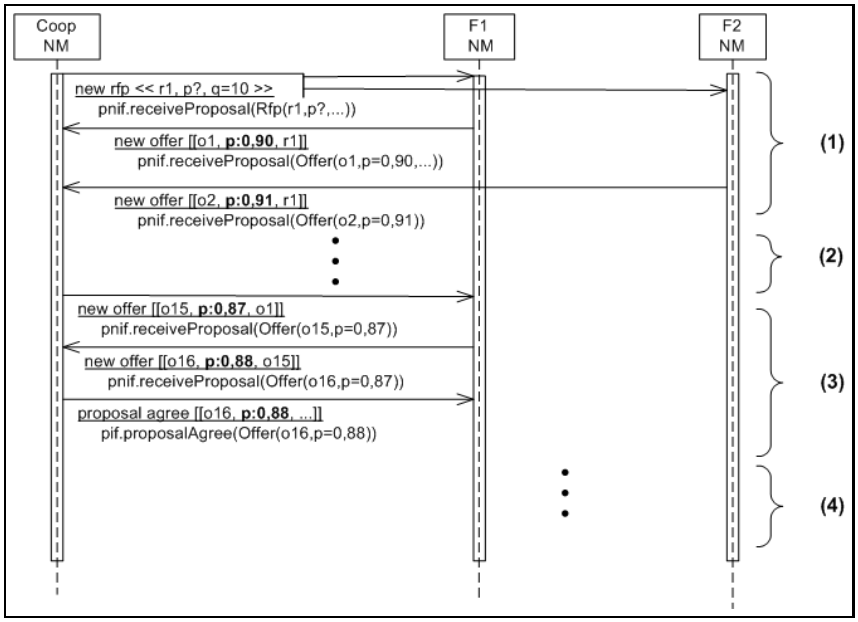


Figure 14: *Quota negotiation*

## 6.1 Supply Chains

Supply Chain Management (SCM) is based on the integration of all activities that add value to customers starting from product design to delivery, in order to minimize system wide cost while satisfying service level requirements [GN04]. It aims at specific processes in order to optimize inventory level, reduce cost and increase profits. The proposal of [KKC04] even goes into budget and payment control. Our approach differs from the usual treatment of SCM in the sense that our goal is to provide generic mechanisms that allow cooperation among the participants of a supply chain without aiming at any specific business process.

Chatfield and others [CHH06] present a supply chain simulation system that is based on a supply chain description, written in SCML - Supply Chain Modeling Language. This description contains structural (*nodes*, *arcs*, *components*, *actors*, and *policies*) and managerial information (properties associated with these constructs, e.g., storage capability, inputs and outputs). Structural and managerial information are used to build on demand a mathematical model that simulates a supply chain.

Our architecture proposes a supply chain structure quite similar to the one proposed by Chatfield. However, it does not contain a separate structure for managerial information, which is instead stated in contracts, regulations,

summaries and coordination plans. This decentralization facilitates local administration of tasks, within Web services, being thus closer to real world supply chains.

## 6.2 Contracts

There are several proposals for e-contract specification. In [ATG05], business contracts specify the exchange of values among business parties and the conditions for the exchange. They require three fundamental classes of language constructs: data constructs, process constructs, and rule constructs. Data constructs define the exchanged values between parties, such as quantities, prices, deadlines, and quality categories. Rule constructs express the conditions for product exchange. Process constructs are responsible for describing the steps of contract enactment.

[LMC+04] models an enterprise as a series of interrelated communities, whose members perform *roles*. Their contract monitoring language (BCL) is made up from the following entities: a) community expressions; b) policies; c) temporal constraints; d) event matching constraints; e) state conditions. In [KKC04], the entities of an e-contract are: parties, clauses, budget, roles and payments. Contracts can be composed into more complex ones. [HLGA01] includes details of the infrastructure needed to carry out contracts.

Our contracts are semi-structured documents composed of clauses. Each clause refers to a duty or a right of a partner. A clause has one or more properties that qualify and quantify such right or dutys, and a number of constraints (e.g., temporal, spatial, quality). Properties must be understood by all the negotiators, using ontologies.

In [WH02], contracts are specified using the XBLC language. An XBLC contract is composed of one or more workflows that specify and coordinate transactions to be run. This kind of contract is used to coordinate and control the interaction between business workflows. This approach is different from ours. Our contracts do not coordinate the activities that must be performed to fulfill the agreement. Coordination is performed by other entities in our architecture [BMM04], called "Coordination Managers". Thus, we separate the obligations (what) from how they can be fulfilled. Again, this is closer to reality, since conditions may change in a supply chain. Commitments are still valid, but the way they can be fulfilled may need to be changed.

Fantinato and others [FdTdSG06] address contracts for e-services supply and consumption based on feature modeling. The approach is similar to ours, in the sense that contracts are generated from existing templates and that their features can be broadly compared with our properties. However, our approach is more general. In particular, while e-services are central in their

work, we use e-services just as a mechanism to start clause execution.

## 6.3 The Negotiation Process

There are a number of mechanisms that guide the negotiation process. Bartolini [BPJ04] constructs negotiation templates that specify different negotiation parameters that can be constrained or open. Chiu [CCH+05] also uses contract templates as a reference document for negotiation. Contract clauses contain template variables, whose values are to be negotiated separately or together (e.g., quantity, delivery date and price). Similarly, [RWG01] uses a contract template that describes the negotiation parameters, how they are interrelated, along with meta-level rules about the negotiation. In contrast, [HCWN03] uses a set of examples of good agreements and it is up to the negotiator to try to get as close as possible to one of the examples.

Others – e.g., [HLGA01] or [NSDM04] – do not present a proper negotiation process. Their approach is based on matchmaking. The approach of [HLGA01] is based on predefined contract templates. The seller creates a Contract Advertising Template (CAT) and submits it to a Matchmaking Engine. The consumer creates a Contract Search Template (CST), and submits his proposal to the same engine. Next, the engine tries to match CATs and CSTs. [NSDM04] proposes a matchmaking facilitator based on description logics that ranks matches within categories.

Like most of these papers, our e-negotiation process is guided by a contract model with properties not yet bound to a value. Thus, broadly speaking, the negotiation process concerns determining values for unbound properties. Like [BPJ04], properties may be open or constrained. We distinguish moreover between optional and non-optional properties.

Negotiation strategy is another issue addressed in the literature. According to [GDtHO01], techniques for designing negotiation strategies can be classified into three categories: (i) game-theoretic, (ii) heuristic, and (iii) argumentation. The first approach models a negotiation situation as a game and attempts to find dominant strategies for each participant by applying game theory techniques. In heuristic-based approaches, a strategy consists of a family of tactics (i.e., a method for generating counter-offers), and a set of rules for selecting a particular tactic depending on the negotiation stage. Argumentation-based approaches extend heuristic ones by introducing issues such as threats (e.g., "that is my last offer"), rewards, etc.

Our negotiators may be implemented using any kind of negotiation strategy. They only need to agree on the negotiation protocol and to have a common vocabulary.

Another issue comprises the languages used to specify the negotiation

rules, the strategy and the language used to conduct the negotiation process itself. In general, languages for rules and strategies are declarative and taken from the AI field, such as Defeasible Logic [GDtHO01] and Courteous Logic Programming [Gro97]. Conversely, languages for the negotiation process are similar to protocols used in distributed systems. Any of those languages faces the problem of heterogeneity of vocabulary and concepts. Although some concepts are well-defined in a negotiation framework (e.g., the negotiation protocol is formally defined), contract variables, such as product names, measure units, and currency, may not be standardized and such differences must be reconciliated on the fly. According to [MPO06], this is aggravated in the dynamic environment of e-commerce negotiations where transactions involve interactions among different enterprises, using different representations and terminologies. To solve this, [MPO06] combine the use of ontologies and agent technologies, in negotiations within car assembly supply chains. Our use of ontologies addresses the same interoperability issues.

Our negotiation process is developed through the exchange of messages among the negotiators that comply with a specific protocol. This is a common approach, like the ones based on FIPA's standards [FIP00], e.g., [MPO06].

Governatori and others [GDtHO01] have a different approach. They propose a negotiation process that uses Defeasible Logic. Each negotiator has a set of facts and rules. A negotiator makes an offer to another, with a set of public facts calculated previously. A second negotiator uses this set of facts and its own knowledge database to decide if it will accept, decline the offer or make a counter-offer.

Table 4 summarizes the negotiation issues discussed in this section.

| **Characteristic** | **Related Work** | **Our Work** |
|---|---|---|
| Number of negotiators | bargain (1:1), biding (1:many) (e.g., English auction), double auction (many:many) | allows all of them |
| Number of items | single item, a bundle of items | bundle of items (properties) |
| Strategy | game-theoretic, heuristic, argumentation | allows all of them |
| Negotiation protocol | guided by templates, examples, matching offers with proposals | guided by templates |
| | exchange of messages, exchange of knowledge (AI database) | exchange of messages |
| | manual negotiation with successive refinement of feature models | human actors may intervene, but not compulsory |

Table 4: Negotiation issues

### 6.4 Enactment

The enactment phase comprises two main activities: the enactment itself and monitoring for audit purposes. Enactment should enforce a number of constraints and audit must verify if they were actually enforced. In this context, BCL [LMC+04] expresses and monitors conditions in business contracts. In addition, [LMC+04] states that a monitor can access a community specification (that represents a contract), collect events significant to the contract from the participants or the environment and interpret them in order to determine whether the contract is being followed. The authors state that it is possible to think of multiple monitors, each protecting the interests of one of the signatories.

Conversely, our contracts do not have monitoring constraints to verify if they have been fulfilled. Clauses have preconditions and postconditions, but even if they hold, it does not ensure the fulfillment of the contract. However, logs produced by clause execution can be used for monitoring purposes following the process mining approach of [vdAvDH+03, vdAdBvD05]. Governatori and others [GMS06] have a different approach. They aim at checking the compatibility of business processes and business contracts by means of a logic-based formalism.

## 7 Conclusions

The paper presented a framework for contract negotiation in agricultural supply chains, which uses our supply chain model ([BMM04]). The main contributions are: i) a contract model that includes the specification of quality constraints suitable for this kind of chains; ii) the negotiation protocol that produces such a contract; and (iii) the implementation of the framework via Web services. The explicit use of ontologies, exemplified in Sections 3 and 4, combined with Web services, increases interoperability and fosters local independence among business partners.

A contract is composed of a set of clauses, with pre- and postconditions – reflecting business rules. Carrying out a contract means activating some of the clauses individually and in an appropriate order. Clause activation triggers the execution of a Coordination Plan (Section 2) – a business process. The Coordination Plan guarantees the appropriate order of clause execution of a contract. Pre- and postconditions (Regulations), in agricultural chains, concern both administrative (e.g., payment schedule) and quality (e.g., food safety measures) constraints.

Two factors differentiate our work from other proposals. First, our nego-

tiation protocol is based on a generic grammar, and supports the specification of a wide variety of negotiation styles, and their implementation, using just a few negotiation primitives. These primitives are reflected in the messages exchanged among partners. This was illustrated by a few interaction scenarios. Most other proposals are centered on establishing specific protocols for given situations.

Second, though intimately related, our proposal clearly separates business processes from contracts and their negotiation. This allows scenarios where a given business process requires multiple independent contracts and negotiations. It allows moreover situations where a contract may be enacted by more than one business process - for instance, one process may be responsible for fulfilling a part of the contract (e.g., material procurement) and another process for another part (e.g., shipment). Moreover, this lets contracts and processes evolve in a transparent way.

This separation between contracts and business processes simplifies process management and supports several real life situations in a flexible way. Contracts define rights and obligations, while processes express how these rights and obligations will be satisfied. Note that any kind of environmental alteration (e.g., new law, natural disaster, internal modification) may demand a change in the way a business process is handled, but may not modify the rights and obligations. That is, they must be fulfilled, but in a different way.

Since negotiation may happen among many negotiators, several contracts may be needed at a given chain stage. This raises a number of interesting problems involving regulation enforcement and audit, especially when renegotiation is allowed, either in the enactment phase or in the renegotiation phase itself. These are themes for future work. Our negotiation protocol allows several styles of negotiation. Thus, future work also includes identifying and describing them in a systematic way.

# References


[Ars02] A. Arsanjani. Developing and Integrating Enterprise Componentes and Services. *Communications of the ACM*, 45(10):31–34, 2002.

[ATG05] S. Angelov, S. Till, and P.W.P.J. Grefen. Dynamic and secure B2B e-contract update management. In *ACM Conference on Electronic Commerce*, pages 19–28, 2005.

[BMM04] 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.

[BPJ04] C. Bartolini, C. Preist, and N.R. Jennings. A software framework for automated negotiation. In *SELMAS*, pages 213–235, 2004.

[CCH+05] D.K.W. Chiu, S.C. Cheung, P.C.K. Hung, S.Y.Y. Chiu, and A.K.K. Chung. Developing e-negotiation support with a meta-modeling approach in a web services environment. *Decision Support Systems*, 40(1):51–69, July 2005.

[CHH06] D.C. Chatfield, T.P. Harrison, and J.C. Hayya. SISCO: An object-oriented supply chain simulation system. *Decision Support Systems*, 42(1):422–434, 2006.

[FdTdSG06] M. Fantinato, M. B. F. de Toledo, and I. M. de S. Gimenes. A feature-based approach to electronic contracts. In *CEC/EEE'06*, pages 34–41, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[FIP00] FIPA. Fipa abstract architecture specification. Available at www.fipa.org, 2000.

[GDtHO01] G. Governatori, M. Dumas, A.H.M. ter Hofstede, and P. Oaks. A formal approach to protocols and strategies for (legal) negotiation. In *ICAIL*, pages 168–177, 2001.

[GM82] H. Garcia-Molina. Elections in distributed computing systems. *IEEE Transactions on Computers*, C-31(1):48–59, jan 1982.

[GMS06] G. Governatori, Z. Milosevic, and S. Sadiq. Compliance checking between business processes and business contracts. In *Proc. 10th Intl. Enterprise Distributed Object Computing Conference*, pages 221–232, 2006.

[GN04] A. Gunasekaran and E.W.T Ngai. Information systems in supply chain integration and management. *European Journal of Operational Research*, 159:269–295, 2004.

[Gro97] B. Grosof. Courteous logic programs: Prioritized conflict handling for rules. IBM Research Report RC20836, May 1997.

[HCWN03] P. Henderson, S. Crouch, R.J. Walters, and Q. Ni. Comparison of some negotiation algorithms using a tournament-based approach. In *Agent Technologies, Infrastructure, Tools and Applications for E-Services*, volume 2592 of *Lecture Notes in Artificial Intelligence*, pages 137–150. Springer, Jan 2003.

[HLGA01] Y. Hoffner, H. Ludwig, P. Grefen, and K. Aberer. Crossflow: integrating workflow management and electronic commerce. *SIGecom Exch.*, 2(1):1–10, 2001.

[IBM06] IBM. Ws-coordination. Available at http://www.ibm.com/developerworks/library/specification/ws-tx/, 2006.

[KKC04] R. Krishna, K. Karlapalem, and D.K.W. Chiu. An ER$^{ec}$ framework for e-contract modeling, enactment and monitoring. *Data & Knowledge Engineering*, 51(1):31–58, oct 2004.

[KMBM07] A.K. Kondo, C.B. Medeiros, E. Bacarin, and E.R.M. Madeira. Traceability in Food for Supply Chains. In *Proc. 3rd International Conference on Web Information Systems and Technologies (WEBIST)*, pages 121–127, March 2007. Barcelona,Spain.

[LMC+04] P.F. Linington, Z. Milosevic, J. Cole, S. Gibson, S. Kulkarni, and S. Neal. A unified behavioural model and a contract language for extended enterprise. *Data & Knowledge Engineering*, 51(1):5–29, 2004.

[MPO06] A. Malucelli, D. Palzer, and E. Oliveira. Ontology-based services to help solving the heterogeneity problem in e-commerce negotiations. *Electronic Commerce Research and Applications*, 5(1):29–43, 2006.

[MZ02] H. Min and G. Zhou. Supply chain modeling: past, present and future. *Computer & Industrial Engineering*, 43:231–249, July 2002.

[NSDM04] T. Noia, E. Sciascio, F.M. Donini, and M. Mongiello. A system for principled matchmaking in electronic marketplace. *Intl. Journal of Electronic Commerce*, 8:9–37, Summer 2004.

[RWG01] D.M. Reeves, M.P. Wellman, and B.N. Grosof. Automated negotiation from declarative contract descriptions. In *Proc.*

*of the 5th International Conference on Autonomous Agents*, pages 51–58, Canada, 2001. ACM Press.

[vdAdBvD05] W.M.P. van der Aalst, H.T. de Beer, and B.F. van Dongen. Process mining and verification of properties: An approach based on temporal logic. In *OTM Conferences (1)*, pages 130–147, 2005.

[vdAvDH+03] W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow mining: A survey of issues and approaches. *Data & Knowledge Engineering*, 47(2):237–267, 2003.

[WH02] H. Weigand and W. Heuvel. Cross-organizational workflow integration using contracts. *Decision Support Systems*, 33(3):247–265, July 2002.