Negociação Automática de Contratos Multi-laterais em Cadeias Produtivas Agropecuárias

Este exemplar corresponde à redação final da Tese devidamente corrigida e defendida por Evandro Baccarin e aprovada pela Banca Examinadora.

Campinas, 22 de dezembro de 2009.

Edmundo R.M. Madeira (Orientador)

Claudia M. Bauzer Medeiros (Co-orientadora)

Tese apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação.

Substitua pela folha com as assinaturas da banca

Instituto de Computação
Universidade Estadual de Campinas

Negociação Automática de Contratos Multi-laterais em Cadeias Produtivas Agropecuárias

Evandro Baccarin[1]

Dezembro de 2009

[1]Suporte financeiro de: Bolsa do CNPq (processo 143153/2006-4) 12/2006 - 02/2007, CAPES (Estágio de Doutorado no Exterior processo 4635-06-0) 04/2007 - 03/2008.

Resumo

Uma cadeia produtiva agropecuária é constituída por diversos tipos de atores que estabelecem uma rede de relacionamentos bastante complexa. Estes relacionamentos variam de *ad hoc* e de curta duração até altamente estruturado e de longa duração. As cadeias produtivas agropecuárias possuem algumas particularidades, tais como, regulamentação estrita e dependência cultural, e possuem relevância social e econômica. A utilização de contratos é a forma natural para expressar os relacionamentos entre os membros de uma cadeia. Desta forma, contratos e a atividade de negociá-los são de grande importância numa cadeia produtiva. Esta tese propõe um modelo para cadeias produtivas agropecuárias que integra suas principais características, incluindo seus aspectos estruturais e sua dinâmica. Em particular, a tese propõe um formato para contratos multi-laterais e um protocolo de negociação que os constrói. Contratos multi-laterais são importantes neste contexto, pois vários atores de uma cadeia produtiva podem construir alianças que compreendem direitos e obrigações mútuos. Um conjunto de contratos bi-laterais não é adequado para tal propósito. A tese também apresenta uma implementação do protocolo de negociação baseado em serviços Web e numa máquina de workflow (YAWL).

Abstract

An agricultural supply chain comprises several kinds of actors that establish a complex net of relationships. These relationships may range from *ad hoc* and short lasting ones to highly structured and long lasting. This kind of chain has a few particularities like strict regulations and cultural influences, and presents a quite relevant economical and social importance. Contracts are the natural way of expressing relationships among members of a chain. Thus, the contracts and the activity of negotiating them are of major importance within a supply chain. This thesis proposes a model for agricultural supply chains that integrates seamlessly their main features, including their structure and their dynamics. Specifically, the thesis proposes a multi-party contract format and a negotiation protocol that builds such kind of contracts. Multi-party contracts are important in this context because several actors of a supply chain may build alliances comprising mutual rights and obligations. A set of bilateral contracts is not well-fitted for such a purpose. The thesis also presents an implementation of the negotiation protocol that builds on Web services and a workflow engine (YAWL).

Dedicatória

Aos meus pais: um sonho sonhado, talvez, mais por eles.

Agradecimentos

Esta monografia resume o trabalho de alguns anos de pesquisa científica. O tempo de doutoramento, porém, não se resume apenas ao seu conteúdo técnico-científico. Compreende também a experiência de vida adquirida, as pessoas com as quais compartilhei um trecho do caminho da vida, as que seguiram por outras sendas, as que encerraram sua caminhada e deixaram saudades. Não é possível expressá-la em prosa científica, em fórmulas matemáticas ou em linhas de código, pois impregna-se na profundidade do ser humano e, por isso, é intraduzível, é intransferível. Atrevo-me a tentar registrá-la na forma de gratidão àqueles que tanto me ajudaram nestes anos.

Devo começar por meus orientadores: o Prof. Edmundo e a Profa. Claudia. A dedicação e o envolvimento deles foram notáveis. Devo-lhes, também, amadurecimento científico e profissional. Muito que aprendi deles será reproduzido agora que retorno as minhas atividades de ensino, pesquisa e orientação na Universidade de Londrina.

Enorme dívida de gratidão ao pessoal do laboratório, principalmente pela companhia e apoio e, especialmente, o Alan e o Luciano pela ajuda na resolução de certos problemas transcendentais e esotéricos que carinhosamente chamamos "bugs".

I would like to thank all people of TU/e and the friends I met in the Netherlands, specially Prof. Wil van der Aalst for allowing me to join his group. His patience, competence and generosity are remarkable. I miss the people of our "coffee corner". Inesquecíveis são a Ana Karla e a Monica que me ajudaram muito quando cheguei em Eindhoven. I miss all people that shared "our kitchen", specially Hristina and Mohamed. They made me feel at home. Sundays were special days: coffee after the International Mass and lunch with a few friends I made there: Marc, Iana, Jorge, Eugenia. Louis is a bosom friend I left in *het Nederlands: hartelijk dank.*

Fui acolhido pelo Adauto no primeiro ano do curso. Também fui acolhido pelo Tio Ninho e pela Tia Deise ao retornar a Campinas após o "Sanduíche". Além do teto e carinho, pude ouvir aquelas histórias de outrora.

Dívida de gratidão à UEL que me deu a oportunidade de dedicar-me exclusivamente ao doutorado. Especialmente, ao pessoal do departamento: todos absorveram uma carga de trabalho extra por minha ausência. Também às agências de fomento Capes, CNPq e Fapesp que me financiaram diretamente ("Sanduíche") ou indiretamente por meio dos projetos de nosso laboratório que apoiaram (WebMAPS e WebMAPS 2, AgroFlow e BioCORE).

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Esta tese apresenta os resultados da pesquisa desenvolvida no Instituto de Computação (UNICAMP) relativa ao arcabouço SPICA ("espiga" em Latim). SPICA significa "SuPply chain Integration, Coordination, contracting and Auditing framework" que tem por objetivo ser um arcabouço completo para integração de cadeias produtivas agropecuárias.

Uma cadeia produtiva é composta por um conjunto de produtores, vendedores, distribuidores, transportadores e armazéns que atuam nestas atividades no contexto de um produto específico [65]. Uma cadeia produtiva agropecuária possui algumas particularidades, tais como: o fluxo de produtos dentro da cadeia produtiva está sujeito a uma ampla gama de controles; tais cadeias são influenciadas por sua localização, pelo clima de sua região e, até mesmo, sofre influências culturais. A escassez de alimentos devido a desastres naturais ou à falta de infrastrutura adequada pode levar, inclusive, a graves conflitos sociais. Assim, o monitoramento e a facilitação da coordenação e cooperação entre os atores de uma determinada cadeia são aspectos relevantes no gerenciamento de cadeias produtivas agropecuárias. Uma forma de estabelecimento de cooperação entre tais atores e do monitoramento da execução de tal cooperação é a construção de contratos. Portanto, a atividade de negociação de tais contratos também é de grande relevância.

As principais contribuições da tese são:

- um modelo e uma arquitetura para cadeias produtivas agropecuárias;
- um processo de negociação e um formato para contratos multi-laterais que visam estabelecer acordos entre participantes da cadeia.
- uma estratégia de utilização do processo de negociação e do contrato multi-lateral para o estabelecimento de organizações virtuais;
- a implementação de um middleware que suporte o processo de negociação baseado num motor de workflow (YAWL);

A tese está organizada como uma coletânea de artigos da seguinte maneira. O Capítulo 2 descreve o contexto da pesquisa e sumariza as suas contribuições. O Capítulo 3 apresenta o artigo publicado na "12th International Conference on Cooperative Information Systems (COOPIS)" [21]. Este artigo descreve o modelo e a arquitetura para cadeias produtivas, incluindo aspectos estruturais e sua dinâmica. No modelo proposto, uma cadeia produtiva pode ser estruturada hierarquicamente. A dinâmica da cadeia prevê a coordenação das atividades desenvolvidas pelos seus atores, bem como o controle da qualidade dos produtos que transitam pela cadeia produtiva, incluindo o registro deste fluxo para fins de auditoria e rastreabilidade. O Capítulo 4 contem um artigo publicado no "International Journal of Electronic Commerce (IJEC)" [16] e descreve o formato contrato multilateral e o protocolo de negociação. O processo de negociação é dirigido por um modelo de contrato. Os negociadores são serviços Web e tem por objetivo preencher as lacunas do modelo. O artigo apresenta as interfaces que os negociadores devem implementar para participar de uma negociação. O artigo apresentado no Capítulo 5, publicado na "11th International Conference on Enterprise Information Systems (ICEIS)", usa o protocolo de negociação e o contrato multilateral para construir organizações virtuais. Ele mostra como os diferentes estilos de negociação podem ser combinados e executados conjuntamente para tal fim. Mostra um exemplo de uma negociação hierárquica. Neste exemplo, uma cooperativa negocia em nome de várias propriedades rurais. Eventualmente, a cooperativa, antes de concordar com uma oferta recebida, consulta (num segundo nível de negociação) as fazendas a fim de verificar se existe consenso sobre tal oferta. O Capítulo 6 apresenta o último artigo, recentemente submetido ao "International Journal of Cooperative Information Systems (IJCIS)". Ele descreve a implementação do *middleware* de negociação auxiliado por uma máquina de *workflow*. Este artigo é produto do "Estágio de Doutorado no Exterior" desenvolvido pelo candidato na "Technische Universiteit Eindhoven" (Holanda), sob orientação do Prof. Wil van der Aalst. Uma versão estendida deste artigo está disponível na forma de relatório técnico [20]. O Capítulo 7 apresenta algumas observações finais desta tese, além de trabalhos futuros. Finalmente, o Capítulo 8 conclui a tese.

Outros artigos foram publicados no decorrer da realização da pesquisa, não incluídos na tese. Foram feitas investigações preliminares acerca de aspectos de coordenação da cadeia produtiva agropecuária nos estágios iniciais da pesquisa. Resultados parciais estão descritos em [19]. O protocolo de negociação foi simulado por meio de redes de Petri durante o estágio de doutorado no exterior ("sanduíche") desenvolvido pelo candidato. Estes resultados estão publicados em [15].

Capítulo 2

Research Context and Contributions

2.1 Overview

This thesis is organized as a collection of articles which present the results of the research developed during the past few years concerning the SPICA framework. SPICA ("corn ear" in Latin) stands for "SuPply chain Integration, Coordination, contracting and Auditing framework". It aims at providing a comprehensive framework for agricultural supply chains. This chapter depicts the context that encompasses the research, summarizes the results we believe are its main contributions and establishes the links between the articles.

A supply chain is a network of retailers, distributors, transporters, storage facilities and suppliers concerning a specific product [65]. These elements are by their very nature distributed, heterogeneous and autonomous and their relationships are inherently dynamic.

An agricultural supply chain has a few particularities. To start with, the flow of products within a chain is subject to a wide range of controls. Besides the economic and delivery schedule limitations found in business to business (B2B) negotiations, agricultural supply chains are sensitive to geographic location, season, climate, product perishability, and cultural and religious backgrounds. Shortage of supplies may even cause social disruption to the point of generalized famine.

Examples of concerns are, for instance, whether the production process is harmful to the environment or whether it uses genetically modified substances. This requires setting up strict monitoring at all stages, as well as enforcing a large set of rules, which may be product, region or season-sensitive. Two parallel concerns are the quality of the final product, which involves auditing all production and distribution stages, and an efficient logistics to avoid or to relief ill-supplied scenarios.

Another peculiarity is the so-called return flow within such chains, in which the refuse of a given stage of the chain may be recycled and re-enter the chain at another stage.

Recycling is not a problem restricted to agricultural chains, but the constraints imposed on these cycles are. Finally, the number and kinds of actors encountered allow limitless possibilities of chain configurations, and the same kind of raw material may originate a large set of interrelated chains.

As already mentioned, the participants of a supply chain are not isolated, self-sufficient entities, but they, by their very nature, are actors of a dense network of relationships. Some of these relationships will require a strict coordination of these actors´s actions. However, most relationships will be based on cooperation: all actors know in advance what they should do, what they might expect from their counterparts and they trust that their counterparts will behave accordingly. Relationships may even be formed in an *ad hoc* fashion and have a short lasting lifespan. These latter two kinds of relationships may not be imposed by a "central authority", but should stem from an agreement among the partners. Thus, negotiation is also an important issue within an agricultural supply chain.

To sum up, coordination and cooperation are major issues to be fostered in agricultural supply chains. The main contributions proposed in this thesis may be summarized as:

- a model and an architecture for agricultural supply chains, presented in Chapter 3;
- a negotiation process and a multi-party contract format for agreements among the participants of a supply chain, described in Chapter 4;
- a way of using the negotiation process and a contract to build a virtual organization, presented in Chapter 5;
- the implementation of a middleware that supports the negotiation process by means of extending a workflow engine, presented in Chaper 6.

There follows an overview of the research challenges attacked in the thesis, detailing the contributions.

2.2 Contributions 1 and 2 - Model and Architecture for Agricultural Supply Chains

The first contribution is a model for agricultural supply chains. The reasons for modeling supply chains are manifold. First, agronomists have to decide the goal of their research. The understanding of the environment (i.e., the supply chain) their research may benefit would help to direct their efforts and resources [29]. Second, in any large country like Brazil, the government has to decide about how to apply different resources to different regions to develop them or to perform disaster relief actions (if needed). Modeling key supply chains may help to understand better the social conditions and intervene there effectively. Finally, entrepreneurs aim at gaining more money. A fine-tuned supply chain model helps optimizing a few logistics issues. All these three dimensions are deeply intertwined, and are part of the supply chain. Ignoring this would narrow the comprehension of the environment and hinder an effective use of resources.

The literature on economics and agriculture often presents descriptions of specific supply chains, e.g., the rice supply chain. However, such descriptions differ significantly in what the chain elements are, their relationships and the level of details of description. For an example of such a disparity, see [29]. This book was written by researchers of Embrapa (Brazilian Agricultural Research Corporation) and other related research corporations and describes fourteen agricultural supply chains. A typical example of such a disparity can be highlighted by means of the descriptions of the silk supply chain [89] and the manioc supply chain [81]. The former displays a figure containing three quite detailed flowcharts to describe the silk chain under three perspectives, whereas the latter depicts the manioc chain by means of a very generic flowchart containing, basically, a farm, an industrial, a few wholesales and retailers. Agronomists also discuss the use of Information Technology in the context of agricultural supply chains, such as, [82] and [26].

There are countless articles on modeling, simulating and assessing supply chains aiming at optimizations or decision support in the area of Operations Research. Chatfield and others [32] propose a modeling language (Supply Chain Modeling Language – SCML). It focuses on storing supply chain structural and managerial information at different levels to assist supply chain analysis, and decision support. Rappold and Tchernev [76] introduce a special issue on supply chain design. According to them, the proposed models aim at predicting "accurately the behavior and performance of a supply chain under various conditions and violations of underlying business assumptions". This involves problems like product design ([54]), inventory optimization ([28]), vehicle routing ([23]), and supplier selection ([90]). Operations Research is not our focus.

SPICA proposes a model for agricultural supply chains, comprising a few elements, namely: products, production, transportation, storage elements and actors. *Products* are the central entity of a supply chain. A *production element* transforms raw material or intermediate products into other products. *Storage elements* typically are warehouses or any place where products are stored. *Transportation elements* move products from one element to another (including other transportation elements). The chain's dynamics is modeled by means of coordination plans, contracts, regulations and summaries. A coordination plan specifies a sequence of activities to be performed within a supply chain. A contract specifies the relationship between chain's elements. A contract can be established among several chain's elements (i.e., it is a multi-party contract). A regulation imposes restrictions on a product's flow. Finally, summaries are used to track the flow and transformation of production within a chain.

One distinguishing feature of SPICA´s model is that one element can be composed of other elements. This gives rise to a hierarchical organization and allows to approach the supply at different levels of details.

The model has already been used in a few case studies associated with the milk supply chain ([38, 57, 56]). While Kondo [57, 56] concentrates in traceability issues, Figueiredo [38] considers triggers to keep track of regulations within a chain. Both proposals were implemented as prototypes.

The second contribution is an architecture for agricultural supply chains. This architecture comprises architectural blocks for each element in SPICA´s model and also a few managers and repositories, e.g., there is a coordination manager and a coordination plan repository concerning coordination plans.

The model and the architecture are presented in Chapter 3. This article inspired other research activities in our group. First, Alan Nakai defended a master's thesis that proposed a choreographic approach for coordination on agricultural supply chains which is related to Coordination Plans and Coordination Managers [67]. Andréia Kondo defended her master's thesis on management of traceability on food supply chains which concerns Summaries and Summary Managers [56], also summarized in [57]. The original model proposed two kinds of summaries: process and product. Kondo added a new one: the service summary. A service, in this context, is any activity that may influence a product without transforming it, e.g., inappropriate product handling may expose this product to some kind of contamination. Thus, the condition under what the product was manipulated should be registered for future investigation (if needed). Later, Mauricio Figueiredo presented a master's thesis that proposes mechanisms to manage rules that specify the quality of products in supply chains also concerning summaries [38].

Concerning SPICA´s model, coordination issues were investigated in the early research stages. This gave rise to a technical report on using choreography to support collaboration in agricultural supply chains [19]. However, as the research evolved, the research efforts were directed to negotiation issues. Thus, coordination issues were left for future work.

2.3 Contribution 3 - Multi-party Negotiation

A negotiation process is the thesis' third contribution. Virtually all authors agree that negotiation is a process by which entities communicate to reach an agreement on matters of mutual interest [35, 36, 43, 49]. Dang and Huhns [35] highlight that the negotiators are autonomous and try maximizing their individuals utilities. Governatori and others [43] are also concerned with negotiation with legal contents. Electronic negotiation (e-negotiation) is a negotiation supported or conducted by means of the Information Communication Technology [36]. The terms negotiation and e-negotiation are used interchangeably throughout the thesis.

2.3.1 Related Work

A negotiation process involves a number of issues. To begin with, the *number of negotiators* involved typically is one-to-one (*bargaining*), one-to-many (*auction*), or many-to-many (*double-auction*). In bargains, one seller deals with one buyer. Auctions aim at selecting one among several competitors and have a broad range of flavors. For instance, the so-called English or Dutch auctions have many buyers competing for a product sold by an auctioneer. In English auctions, the auctioneer defines a start price and the bidders increase continuously the item´s value until the last bid cannot be beaten. Dutch auctions start with a high value and the auctioneer decreases this value until a buyer agrees to pay the proposed value. A many-to-many negotiation can occur through a *double auction*. In this case, several sellers offer their products on a "shared space" and several buyers submit their bids. There is a mechanism that matches offers and bids.

Even if the negotiation process develops in one-to-many or many-to-many fashions, the final agreement, in general, occurs between exactly two parties: a seller and a buyer. As far as we know, little research has been done concerning negotiation among three or more parties looking for mutual agreement. Bartolini and others ([22]) present a framework for negotiation and advocate that it extends naturally to the case of agreements among multiple parties, but it does not explicitly show how this is achieved. One particular kind of such negotiation, not addressed in any of the consulted articles, is quota negotiation where multiple parties agree to fulfill a specific goal.

A second issue about the negotiation process is related to the number of items. The negotiation process may be restricted to a single product or to a bundle of items. In the latter case, the buyer may be interested in buying all items or none [77]. This kind of negotiation can be developed by *combinatorial auctions* [34].

A third issue refers to the negotiation strategy. According to [43], techniques for designing negotiation strategies can be classified into three categories: (i) game-theoretic, (ii) heuristic, and (iii) argumentation.

Approach (i) 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 (ii), 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 stage of the negotiation. Argumentation-based approaches (iii) extend heuristic ones by introducing communication performatives such as threats (e.g., "that is my last offer"), rewards, etc.

The negotiation strategy depends on a number of factors. The first factor concerns how the negotiation process is conducted. Bartolini and others [22] construct negotiation templates that specify different parameters of negotiation (product type, price, etc) that can be constrained (range of possible values) or open (e.g., price). Chiu and colleagues [33] also use contract templates as a reference document for negotiation. They are composed of contract clauses that contain template variables, whose values are to be negotiated. There may be dependencies among such variables, such that some of them should be negotiated together (e.g., quantity, delivery date and price), and others afterwards. These relationships influence the negotiation plan. Similarly, Reeves and others [77] use a contract template that describes the negotiation parameters, how they are interrelated, along with meta-level rules about the negotiation. In contrast, Henderson and others [50] use a set of examples of good agreements and it is up to the negotiator trying to get as close as possible to one of the examples.

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 strategy are declarative and taken from the AI field, such as Defeasible Logic [43] and Courteous Logic Programming [46]. 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 [61], this is aggravated in the dynamic environment of e-commerce negotiations where transactions involve interactions among different enterprises, using different representations and terminologies.

The fifth issue concerns how the negotiation process is finished: either by agreement or by withdrawal. In [22], agreement formation rules determine which proposals are matched. In [50], negotiation is aborted if agreement is not reached in a number of predefined rounds.

Finally, the last issue concerns how to renegotiate an existing contract due to some external change. This includes discovering which contracts were affected by the change, which clauses should be modified, who should modify the contract, and so forth.

2.3.2 SPICA Negotiation

SPICA's negotiation process combines desirable characteristics, as discuted in related work. It proposes a contract format and a negotiation protocol. In this protocol, the negotiation process is orchestrated by a leader and is guided by a contract model. The *contract model* is a predefined contract template containing a set of so-called *properties* which are filled in with values agreed upon by the negotiators. There is a notary responsible for 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 the negotiation process by means of asynchronous messages. The messages may be peer-to-peer or broadcasted. After a successful negotiation, a new contract is created from the model and the negotiated values.

There are three generic styles of negotiation: ballots, auctions and bargains. Ballots are used when the negotiators have to reach consensus on a property's value. Auctions are used when there is competition among different negotiators in order to bind a property to a value. Bargains are used when there are two negotiators and they want to interact to reach a value that is convenient for both.

The negotiation styles are built on a few negotiation primitives, i.e., types of messages exchanged among the participants. These primitives rely on four basic mechanisms: request for proposals (RFP), offers, Request of Information (RFI) and information (Info).

An RFP is an invitation. A negotiator A sends an RFP to a negotiator B asking for a value for one or more properties. An RFP may prescribe some restrictions on the expected answer and may also bind the value of other properties.

An offer is a promise. A negotiator who wants to assign a value to one or more properties sends an offer to another negotiator. The offer indicates the properties the first negotiator is interested in and the values it proposes for them. If the other negotiator accepts such an offer, both negotiators are committed to the proposed values. A negotiator can answer to an RFP by sending back an offer that proposes values for the desired properties and that complies with the restrictions indicated in the RFP.

RFIs and Infos are similar to RFPs and offers, respectively. However, there are two distinguishing differences. Firstly, besides asking a value for a given property, an RFI may ask upper and lower bounds for it. Secondly, Infos provide the asked information, but the negotiator is not committed to the provided values.

2.4 Contribution 4 - Multi-party Contracts

The contract format is the thesis' fourth contribution. *Contracts* are statements of common intentions which comprise the mutual obligations and authorizations that reflect the (legally binding) agreements between two trading partners [86]. Hanson and Milosevic [49] add that such agreement includes the consumption of resources, fulfilling requirements and the communication of information. An *e-contract* is a contract modeled, specified, executed and enacted (controlled and monitored) by a software system [58]. From now onwards, the terms contract and e-contract are used interchangeably.

2.4.1 Related Work

Contracts can be categorized based on their nature of execution as *sequential*, *cyclic* or *turnkey* [58]. As will be seen, SPICA contracts can follow any of these styles. A sequential contract executes sequentially once. A cyclic contract holds good for a certain period of time regardless of how many times the contract is fulfilled. A turnkey contract has a specific goal that needs to be accomplished within certain time and with a certain budget.

Contracts can also be categorized as bilateral or multi-party contracts. A bilateral contract is signed exactly by two partners whereas a multi-party one may have several signatories. SPICA contracts support both modes.

A contract´s life cycle starts with the building of some kind of electronic document that represents the agreement. This document follows some specification language and can be built automatically through a negotiation process or other means.

According to Angelov and others [11], 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.

In an article by Weigand and Heuvel [86], contracts are specified using a language called XLBC.[1] An XBLC contract is composed of one or more workflows. One workflow, for instance, is responsible for receiving a purchase order from a consumer; another, for performing the product shipment. When one such workflow is executed, transactions are run.

Linington and others ([60]) model an enterprise as a series of interrelated communities. A community is a configuration of objects that interact to achieve some shared objective. The ODP Reference Model provides a framework that integrates support for distribution, interworking, and portability. These objects perform one or more *roles* in the community. Communities can be hierarchically specified. This paper proposes a contract monitoring language (BCL), also presented in [63], developed for the purpose of expressing and monitoring conditions stated in business contracts. BCL is made up from the following entities: a) community expressions; b) policies (in essence they express fragments of behaviour that need to be monitored to determine the parties' compliance to the contract);

[1]XBLC is named after FBLC, which stands for Formal Language for Business Communication.

c) temporal constraints; d) event matching constraints; e) state conditions.

Moreover, in [58], the entities of an e-contract are: parties, clauses, budget, roles and payments; Hoffner and others [51] include details of the infrastructure needed to execute contracts; and Krishna and other [58] allow composition of contracts (subcontracts).

After being built, the contract may be stored in some kind of repository. Besides storage and searching functionalities, such repositories may provide more specialized functions. For instance, Angelov, Till and Grefen,[11] use a trusted party called e-Notary to store contracts and to verify the authenticity of the contracting parties.

A contract´s life cycle also includes the enactment phase. In this phase, the contract produces the planned effects and its execution is monitored in order to check if the parties have been properly accomplishing the agreed duties. The way a contract is executed and monitored depends on the semantics of its specification language and on the underlying infrastructure.

Contracts are also used in [51] as a means for integrating workflows of different organizations. It uses the contract´s details for dynamic construction of the infrastructure needed for the enactment of the service. For each contract is created and configured a module called Integration Facilitator (IF) in each partner (consumer and provider). The configuration is derived from the contract itself and from a specification called Internal Enactment Specification (IES), that describes the manner by which this specific contract is to be implemented in the respective organisation.

In BCL [63, 60], expected behaviour is defined by *behaviour expressions* associated with a role. Such expression is typically an event pattern. This specification is submitted to the Basic Activity Monitoring (BAM) engine. BAM engine responds to the events as they occur, thus monitoring the execution of business tasks.

Monitoring activities do not suffice. Signatories may be interested on assuring that contract´s provisions will be fulfilled. Milosevic and others define *contract enforcement* as "a set of mechanisms that ensure that contracting parties' behaviour complies with the behaviour specified in the agreed contract" [64]. They identify two types of enforcement. The first, *non-discretionary enforcement*, uses mechanisms that prevent contract breaches. Conversely, *discretionary enforcement* provides control mechanisms aiming at mending a breached contract. They also propose a mechanism congruent with the latter approach that comprises a mediator and an arbitrator. The mediator tries to settle the dispute between the conflicting partners. In case it fails, the arbitrator can apply penalties.

Conditions may change during contract validity, thus the contract has to be updated, either entirely (a new version replaces the old one) or through addenda (modifications stated in separate documents linked to the existing contract).

Angelov and others [11] categorize contract updates according to their predictability and protocols required for performing the update. In terms of predictability, changes can be *anticipated* (changes are foreseen when a contract is established), or they are *exception* updates.

Protocols can consider: *free updates* that follow no constraints (e.g., change of a URL); *rule governed updates* based on rules defined in the contract (e.g., automatic price increase according to inflation rate); *acknowledgment updates* that need explicit acknowledgment from the counter-party; and *renegotiation updates*, requiring multi-step negotiation for contract update.

Electronic contracts are usually digitally signed. Thus, any contract change incurs in the necessity of re-signing the modified contract by all the involved parties. Update and enactment conditions have repercussion on management of *e-contract* repositories – e.g., the proposal from Angelov et al [11] where e-Notaries check contract signatures.

Service Level Agreements (SLA) can also be considered contracts. However, they have a very specific purpose: defining values to predefined quality attributes of a service available to users. They are, in fact, only an instance within our negotiation framework.

2.4.2 SPICA Contracts

A SPICA contract is an XML document organized into a few sections, similar to [86]. Its main section contains a set of clauses. A clause states some kind of obligation and a set of partners involved in fulfilling such an obligation. There might be more than only two partners. In addition, different clauses may have a different set of involved partners. A clause also states how the involved partners are expected to fulfill such an obligation, e.g., a clause may specify that it will be fulfilled if only one partner executes the intended action, other clause might demand that all involved partners should execute the intended action. This arrangement makes SPICA´s contracts actual multi-party contracts.

The contract format and the negotiation process are described in the second paper of the thesis. It is included in Chapter 4. This paper also addresses quota negotiation. A few details were added in [18]. The negotiation protocol was simulated by means of Petri nets during the candidate's stay in the Technical University of Eindhoven under the supervision of Prof. Wil van der Aalst. They are presented in [15].

2.5 Contribution 5 - Using Contracts to Build Virtual Organizations

The fifth contribution of this thesis is in Chapter 5. This article presents a way of using SPICA's multi-party contracts and combining bargains, auctions and ballots to assemble a virtual organization. Virtual Organizations (VO) are dynamic alliances of enterprises that together can take advantage of economies of scale when available [83]. Grefen and others ([45]) push this dynamic aspect to the limit, stating that new virtual organizations (virtual enterprise, in their parlance) should be assembled in a span of a few days or even lesser. They call such an organisation *instant virtual enterprise*.

Chapter 5 basically makes two proposals: the contract format we propose in this thesis, being multi-party, is suitable to describe virtual organisations; the negotiation protocol we propose is flexible and can be used to assemble virtual organisations by means of negotiating a contract among the VO's partners.

2.6 Contribution 6 - Implementation

The sixth contribution is the implementation of the negotiation framework over a tailor-made middleware. It is described in an article submitted to the International Journal of Cooperative Information Systems (IJCIS) included in Chapter 6. This implementation comprises around 330 Java classes. The middleware builds on a workflow engine (YAWL). The middleware, shown in Fig. 2.1, aims at facilitating the implementation of different kinds of negotiators. The negotiation protocol is described by means of a few workflows (e.g., there is a workflow for handling auctions). The flow of messages enables the workflow´s tasks that are automatically executed by the workflow engine. Basically, the execution of a task either dispatches a negotiation message to the intended receiver or uploads a message from a negotiator into the workflow.

The choice of YAWL is based on a few factors. First, YAWL builds on Petri nets. YAWL provides an editor to design workflows. This editor provides an analysis tool that can spot a few design problems, such as, potential deadlock situations. YAWL extends Petri nets: workflow specifications may define cancellation regions which help handling, in our case, time out situations (e.g., in auctions). Second, YAWL provides a workflow engine that frees us from taking care of low level synchronization problems (e.g., determining when an OR-join is enabled). Third, YAWL is easily extended by means of Web services. YAWL provides a standard mechanism to attach an external routine (the so-called *Custom Service*) to a task. Such a routine is a Web service that is invoked automatically by the engine when its respective task is enabled. This feature allowed us to implement a standard Custom Service (the so-called *NS Custom Service*) that is attached to most of the protocol´s tasks. It validates negotiation messages only considering the task´s input parameters, i.e., the workflow describing a negotiation style (e.g., an auction) can be changed without rewriting NS Custom Service.

It must be pointed out that the implementation did not cover all aspects specified. For instance, Chapter 4, points out that we adopt Molina´s leader election proposal [42]. In our implementation of this aspect, the leader is imposed without loss of generality.

Consider again Fig. 2.1. There is a negotiator (Negotiator 1) willing to make an offer to another negotiator. Chapter 4 presents the interfaces a negotiator should implement to take part in negotiations (represented by small circles in this figure). Thus, Negotiator 1 should invoke the operation `receiveProposal` available at a specific interface (namely, PeerNegotiation interface) implemented by the other negotiator. The implementation effort would be helped if the negotiator partner was a local Java object and such an operation were only a method call. Unfortunately, it is not the case: the partner is not local and a negotiation message conveys extra information needed for house keeping (e.g., to correlate messages in a conversation, to detect if a message has aged, to name but one factor). A communication adaptor (`Com Adp`, in the figure) is used to spare the negotiator of having to be aware of such chores. It mimics the negotiator. Thus, Negotiator 1 only makes a local invocation to its associated adaptor which effectively creates a negotiation message (an XML document) and delivers it to the middleware by means of a `checkIn` operation provided by the middleware. The middleware validates and transports the message and delivers it to a dispatcher. The dispatcher unfolds the negotiation message and invokes the appropriate operation at the receiver (`receiveProposal`, in the example). If needed, the dispatcher broadcasts the message to several receivers.

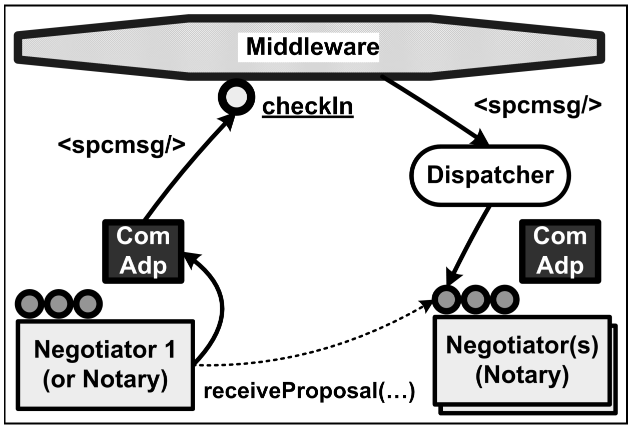
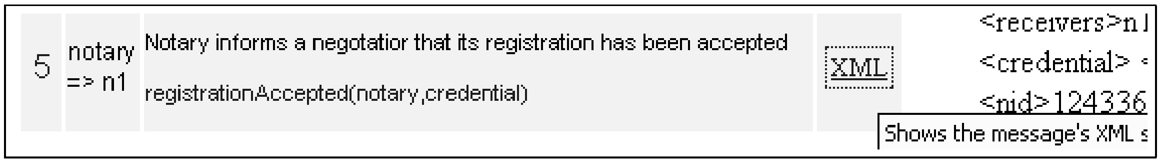


Figura 2.1: *Middleware: basic arrangement.*

This arrangement was implemented incrementally. The first prototype comprised only local Java objects that played the scenario described above. The middleware was a simple object: it only forwarded incoming messages to a local dispatcher that invoked methods of local negotiators. A few auxiliary classes were implemented in this setup. One of them is a logging facility class. It logs all messages conveyed in the middleware and produces a HTML report displaying all messages for demonstration purposes. One log line is shown in Fig. 2.2. The logged message (5th in the log) was sent by the `notary` to a negotiator named `n1`. The third column shows a short explanation about the message semantics and the invoked operation. The fourth column has a link to the serialized message. Finally, the last column shows a part of a serialized message.



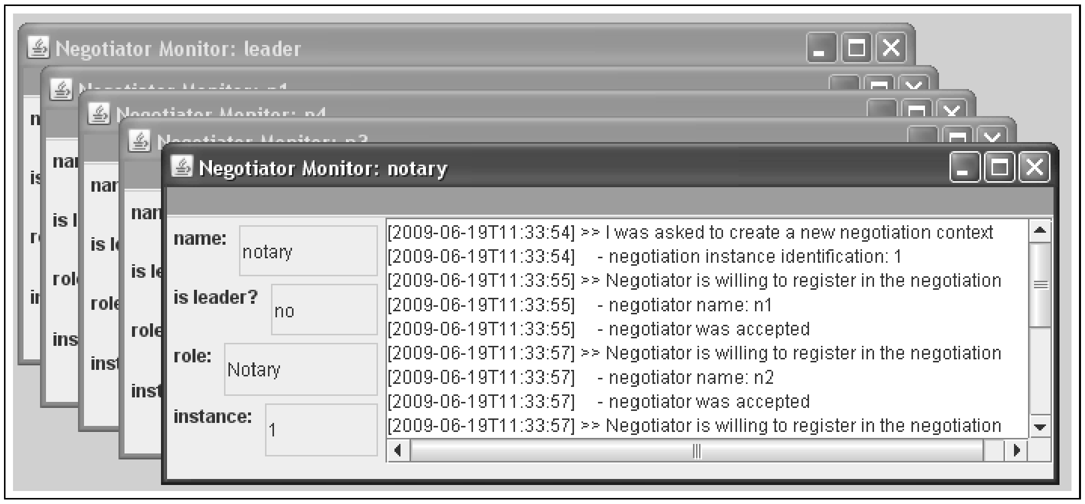
5	notary => n1	Notary informs a negotatior that its registration has been accepted registrationAccepted(notary,credential)	XML	<receivers>n1 <credential> <nid>124336

Shows the message's XML s

Figura 2.2: *Message logging.*

Messages are logged in the communication adaptor because such an adaptor knows at a specific point in time the invoked operation, the serialized message, the sender and the receiver. In the first prototype, all messages pass through the same adaptor, making it easy to keep the message order.

Another auxiliary class is a kind of a dashboard the negotiators and the notary use to inform the actions they decided to perform. In future versions, a negotiator (or notary) will use the dashboard to request the assistance of a human operator about an action to take (e.g., if it should agree on a specific offer). Figure 2.3 shows the five dashboards: four negotiators and the notary.



name: notary
is leader? no
role: Notary
instance: 1

```
[2009-06-19T11:33:54] >> I was asked to create a new negotiation context
[2009-06-19T11:33:54]    - negotiation instance identification: 1
[2009-06-19T11:33:55] >> Negotiator is willing to register in the negotiation
[2009-06-19T11:33:55]    - negotiator name: n1
[2009-06-19T11:33:55]    - negotiator was accepted
[2009-06-19T11:33:57] >> Negotiator is willing to register in the negotiation
[2009-06-19T11:33:57]    - negotiator name: n2
[2009-06-19T11:33:57]    - negotiator was accepted
[2009-06-19T11:33:57] >> Negotiator is willing to register in the negotiation
```

Figura 2.3: *Negotiation Console.*

The second prototype distributed all negotiators and notary. They were wrapped by a web services layer. The dispatcher invokes specific operations at this layer that delegates the execution to the wrapped negotiator (or notary). Their implementation was not changed at all, only the communication adaptor was adjusted to invoke the middleware's `checkIn` by means of a web service interaction. Now, each negotiator has its own instance of a communication adaptor. Thus, the message order cannot be guaranteed anymore. The logging facility was also transformed into a webservice. In this way, a communication adaptor sends log lines to it by means of web services operations.

The middleware's implementation in the first and second prototypes are simple. They only transport an incoming message. Messages are not ckecked. They are believed to be well-formed and to have been sent in the proper order and timing. The third prototype addresses the correctness of incoming messages. It uses a workflow engine (YAWL) to keep the current state of a negotation instance. This allows for, e.g., checking if an arrived message was indeed expected or if it has not aged.

2.7 Thesis Organization

The following chapters contain the articles that are part of this thesis, as follows. Chapter 3 is the paper "A Collaborative Model for Agricultural Supply Chains" presented in the"12th International Conference on Cooperative Information Systems (COOPIS)" [21]. Chapter 4 is the paper "Contract E-Negotiation in Agricultural Supply Chains" published by the "International Journal of Electronic Commerce (IJEC)". Chapter 5 presents "Assembling and Managing Virtual Organizations out of Multi-party Contracts" published in the "11th International Conference on Enterprise Information Systems (ICEIS)". Chapter 6 presents the last paper – "SPICA's Multi-party Negotiation Protocol: Implementation using YAWL", submitted to the "International Journal of Cooperative Information Systems (IJCIS)". Finally, Section 7 discusses the experiences gathered in the research and lists future work.

Other papers were published, but were not included in this thesis, namely: [19], [15], and [20].

Capítulo 3

A Model for Agricultural Supply Chains

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

Abstract
This paper presents a collaborative model for agricultural supply chains that supports negotiation, renegotiation, coordination and documentation mechanisms, adapted to situations found in this kind of supply chain – such as return flows and composite regulations. This model comprises basic building blocks and elements to support a chain's dynamic execution. The model is supported by an architecture where chain elements are mapped to Web Services and their dynamics to service orchestration. Model and architecture are motivated by a real case study, for dairy supply chains.

3.1 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 [59, 65]. It is composed of distributed, heterogeneous and autonomous elements, whose relationships are dynamic, and change while the chain is activated. 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 [12].

This paper is concerned with modeling, supervising and coordinating processes in agricultural supply chains, a specific kind of chain that has a large economic impact all over the world. These chains present new challenges in their specification and management, which so far have been mostly ignored by Computer Science researchers.

To start with, the flow within a chain is subject to a wide range of controls. Besides the economic and delivery schedule limitations found in B2B negotiations, agricultural supply chains are sensitive to geographic location, season, climate and product perishability. Examples of concerns are, for instance, whether the production process is harmful to the environment or whether it uses genetically modified substances. This requires setting up strict monitoring at all stages, as well as enforcing a large set of rules, which may be product, region or season-sensitive. A parallel concern is the quality of the final product, which involves auditing all production and distribution stages.

Another peculiarity is the so-called "return flow" within such chains, in which the refuse of a given stage of the chain may be recycled and re-enter the chain at another stage. Recycling is not a problem restricted to agricultural chains, but the constraints imposed on these cycles are. Finally, the number and kinds of actors encountered allow limitless possibilities of chain configurations, and the same kind of raw material may originate a large set of interrelated chains.

Our solution combines research in databases, computer networks, and distributed systems and is based on tackling the problem in several stages and levels. The first stage involves *modeling the chain's components* and *dynamics*. Subsequent stages consist in *mapping* the chain to our architecture, whose elements are seen as Web Services.

For each of these stages, the chain's elements and flow have to be considered at two levels: within and across enterprises. Furthermore, service coordination also considers two levels: global dynamics, treated by Coordination Plans; and inter-element dynamics, treated via Contracts negotiated between trading partners.

The main contributions are the following: (i) a general model for specification of agricultural supply chains, which takes into consideration cross organizational collaboration aspects; (ii) an architecture for its implementation, which emphasizes coordination and service flow composition issues; (iii) the validation of the model via a real life case study in agriculture, stressing the peculiarities of this kind of application domain.

The rest of this paper is organized as follows. Section 3.2 provides an example that will be used throughout the paper to illustrate our work. Section 3.3 describes the model. Section 3.4 specifies the architecture and shows how it supports dynamic behaviour. Section 3.5 outlines an implementation of a chain via Web services. Section 3.6 contains related work and section 3.7 concludes the paper.

3.2 Agricultural Supply Chains

This section presents a simple agriculture supply chain that will be used throughout the paper to illustrate our solution. Figure 3.1 shows this example – the *dairy cattle supply chain*. The goal of this dairy chain is to process milk, producing and commercializing its products – such as bottled milk, butter, or cheese. The starting point is a "Milk Producer" – a farm that has milk-cows. The farmer gathers milk at given periods. Next, milk is delivered by some sort of transportation means "Transport 1" to a Dairy (production). It can only be processed if it obeys certain constraints stated in "Regulation 1". At the "Dairy", it is processed to create products, which are then transported for 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: lawyers, commodity brokers, quality certifiers or software agents.

Some of the chain´s refuse may provide feedback to it, in terms of return flows – such as from the Dairy back to the Producer. For instance, milk that overflows from vats returns to the farms to be used in cattle feed.

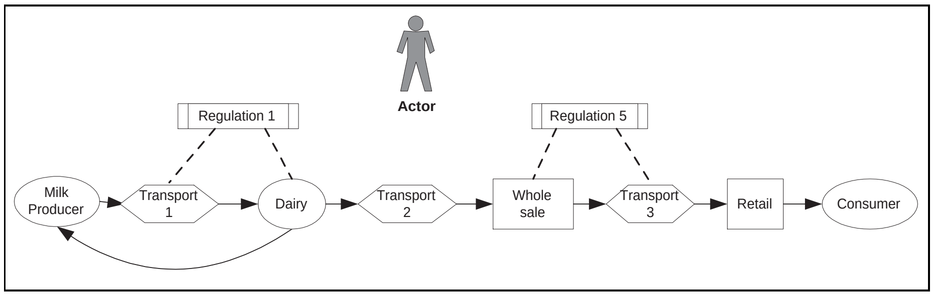


Figura 3.1: *The Dairy Supply Chain*

Even though the diagram in Fig. 3.1 shows a sequential execution, this is seldom the case. Each chain component may moreover encapsulate other chains. Negotiation, cooperation and coordination issues occur at all levels. Coordination may be centralized – such as in the milk cooperative – or distributed among several coordination centers, that negotiate with each other.

3.3 A Model for Supply Chains

3.3.1 Basic Elements

The model's basic elements are Actors, Production, Storage and Transportation. Chain dynamics are furthermore supported by elements Regulation, Contract, Coordination Plan and Summary.

A *Production Element* encapsulates a productive process that uses raw material extracted from its own environment or inputs obtained from other components and produces a product that is passed on 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.

Actors are software or human agents that act in the chain. They may be directly or indirectly involved in the execution of activities A *Regulation Certifier* is an actor that is responsible for certifying that activities or products within the chain obey a set of constraints – such as sanitary regulations or quality specifications.

Regulations are sets of rules that regulate a product's evolution within the chain. These rules specify constraints imposed at distinct execution stages, such as government regulations, quality criteria, or conditions determined by a region's social, cultural, economic or even religious context. Regulations may be atomic or complex, containing other regulations within them.

Interactions among chain components are organized by means of *Coordination plans* and negotiated via *Contracts*. A *Coordination plan* is a set of directives that describe a plan to execute the chain. A chain is coordinated by a top level plan, which may furthermore activate other plans. 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.

Contracts are statements of shared purpose which comprise the mutual obligations and authorizations that reflect the agreements between trading partners [86] that define quality, delivery schedule and costs.

Summaries are elements introduced for traceability and auditability. They are similar to logs, recording chain execution, and may be of two kinds: process and product summaries. A *process summary* contains information about the execution of a production process. A *product summary* stores information on how, when and where a product went through each chain step. It also includes information on certification "stamps" received throughout chain execution.

Dynamics and execution depend on coordination plans, which specify valid element interactions in a very high level. During execution of a specific chain instance, elements are instantiated, contracts negotiated, and the Coordination plan is refined. A Coordination Plan is completely specified only at the end of the execution of a chain, since real-time contract negotiations will dynamically change the chain's configuration, as well as the partners involved.

3.3.2 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.

The degree of composition of the elements depends on the level of detail desired. Figure 3.2 shows how the Dairy Production element of Fig. 3.1 can encapsulate other production chains. Composition and encapsulation of other elements can be likewise exemplified. Raw milk that arrives at the dairy is pasteurized and stored at the "Milk Warehouse". It may subsequently be bottled within the "Bottling of milk" production element, or be transported via the "Transport 6" element to the "Cheese Production" element.

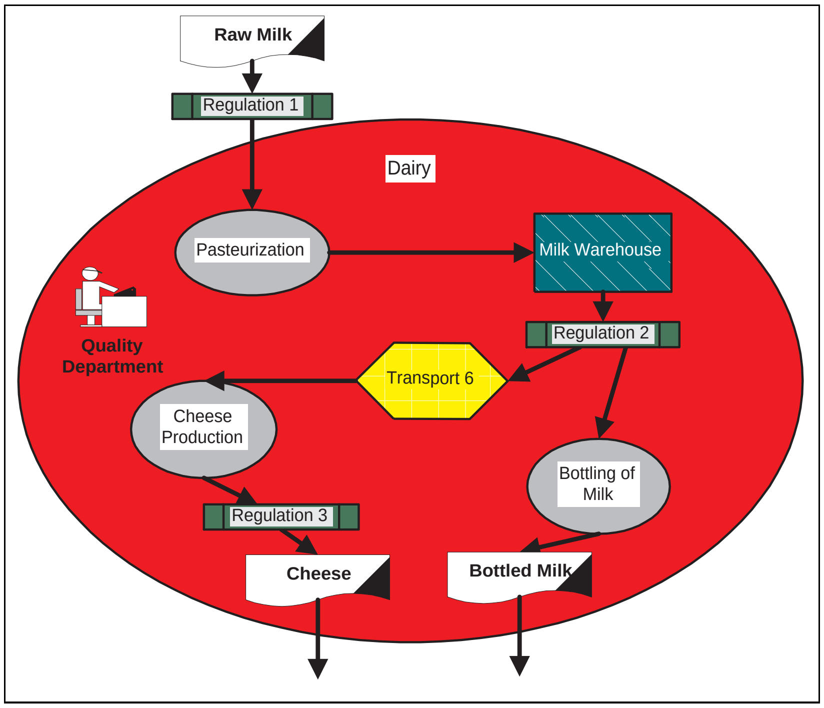


Figura 3.2: *Breaking down Dairy production element*

The placement of a Regulation element within a chain indicates when and where it is applied. "Regulation 1" represents conditions established by the Dairy to accept Raw Milk. They include parameters such as: milk acidity or fat content as well as milk region provenance - e.g., for sanitary reasons. Thus, it is location-sensitive. "Regulation 2" defines rules that determine whether the milk is suitable for cheese or bottling and "Regulation 3" represents quality conditions for cheese comercialization.

Actor "Quality Department" is a sector within the Dairy to check some of the conditions expressed within Regulations 2 and 3. It is within the Dairy element denoting that it can only enforce regulations within it.

3.3.3 Return Flows

Most supply chain studies ignore return flows, unless they model products returned by a consumer. Waste reuse is seldom considered. Environmental concerns are forcing producers to consider residues. Thus, harmful waste is now being returned to its producer or reprocessed, creating *return flows* in the supply chain. Return flow constraints are modeled within regulations and the flow is modeled by backward or forward links between a chain's components.

3.4 The Architecture

3.4.1 Building Blocks

The architecture supports the model described in section 3.3. It is composed of blocks that encapsulate data and/or services. These blocks can be classified into: those that represent the model's basic elements; those used to support coordination, negotiation, documentation and regulation enforcement; and those used for data needed for a chain's execution and auditing.

The basic elements of our model are directly mapped to the architecture's blocks Production, Storage, Transport and Actor. Manager (M) blocks are introduced in order to handle chain dynamics. 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 section 3.3. Furthermore, distinct kinds of repositories are needed to store information on: chain Participants (the basic elements), Products, Regulations, Contracts and Summaries. The contents and roles of Production, Transport, Storage and Actor blocks are straightforward. There follows a description of manager and repository blocks.

Repository Blocks.

Information about chains' elements and execution is stored in six kinds of repositories. Any implementation of the architecture requires that there be at least one repository of each kind, under the responsibility of specific managers.

A *Participant repository* stores cadastral data on a chain's basic participants, namely: Transportation, Storage, Production and Actors. Its goal is to allow validation of the identity of the agents acting within the chain, as well as the roles played by them. It also helps the process of chain instantiation, by supporting the selection of actual businesses to play a given role within a chain.

A *Product repository* contains data on all products and materials used within a supply chain. Its goal is to allow verification of product properties, as well as supporting cross-references within and across chains.

A *Regulation repository* stores regulations for contract negotiation and quality control. Such regulations include global rules (e.g., government level) and local rules (e.g. within a production process).

A *Contract repository* stores contracts established among chain components. More details on these contracts are provided in the next section. A *Coordination plan repository* contains coordination plans specified at distinct granularity levels. The coordination plan repository also contains information about plan execution (e.g., instantiation, validity).

All these repositories support composition of their elements. Thus, composite contracts can be built by aggregating other contracts, plans can be built from the composition of previously stored plans, and so on. Summaries, on the other hand, record the execution of a chain and thus cannot be created from past summaries.

A *Summary Repository* stores product and process summaries, for documentation and auditing. Thus, they can be controlled by government agencies, such as health or sanitation departments, to check on the quality of products and of the production process.

Manager Blocks.

The chain's elements and flow have to be considered at two levels: within and across enterprises. Furthermore, service coordination also considers two levels: global dynamics, treated by a Coordination Plan; and inter-element dynamics, treated via the negotiation of Contracts between trading partners. 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.

A *Coordination manager* is in charge of Coordination Plans, interpreting, controling and coordinating them. It is also responsible for managing the Coordination plan Repository. Therefore, these managers trigger and coordinate all processes within the chain. In particular, they are responsible for starting negotiation among components, and may also start regulation enforcement procedures.

A *Negotiation manager* is responsible for handling contracts and coordinating negotiation among distinct chain elements. It also controls Contract Repositories.

A *Summary manager* controls access to a Summary Repository. A *Regulation Manager* encapsulates the access to a Regulation Repository and is also used to verify regulations using information from all repositories. It informs to the Coordination and Negotiation managers whether a regulation has been obeyed or not. Thus, it does not play an active role in regulation enforcement.

3.4.2 Orchestration of the Supply Chain

The backbone of all orchestration interactions within a chain is formed by a hierarchy of Coordination Managers, that communicate along specific protocols based on a coordination plan. A coordination manager CM at a given hierachical level can only communicate with its parent and its children (levels immediately above and below).

All other interactions among managers are described in terms of this coordination hierarchy background. Each coordination manager CM in the hierarchy may be associated with at most one regulation manager RM, one summary manager SM and one negotiation manager NM. These three managers (RM, SM and NM) are said to be *within the scope* of that coordination manager.

A coordination manager, furthermore, interacts with: the negotiation and summary managers within its scope; and with all regulation managers above its level, and the regulation manager within its scope.

Consider again the "Milk Producer" and "Dairy" elements of figure 3.1. Suppose that the milk producer is, in fact, a cooperative that agregates several milk farms and the dairy is composed of three production units (for butter, bottled milk and cheese). Figure 3.3 depicts the block arrangement for those elements and some of their interactions. This example details only production elements, but similar arrangements may also be done for transportation or storage elements. The figure shows a 2-level hierarchy, rooted at CM3.

NM1, RM1 and SM1 are within the scope of CM1 (the cooperative's coordination manager). CM1 can communicate with CM3 (its parent in the coordination hierarchy), with the farms (its children), NM1, RM1 and SM1 (the managers within its scope).

A Negotiation Manager can interact with any other negotiation manager, and with regulation managers of the same scope or above. Negotiation is always triggered by a coordination manager interacting with a negotiation manager.

A Regulation Manager may interact with regulation managers at any level above it. They may also respond to requests from any negotiation manager within the same scope or below its level, and to the coordination manager within the same scope or below its level.

Summary Managers only interact with coordination managers and with any other SM.

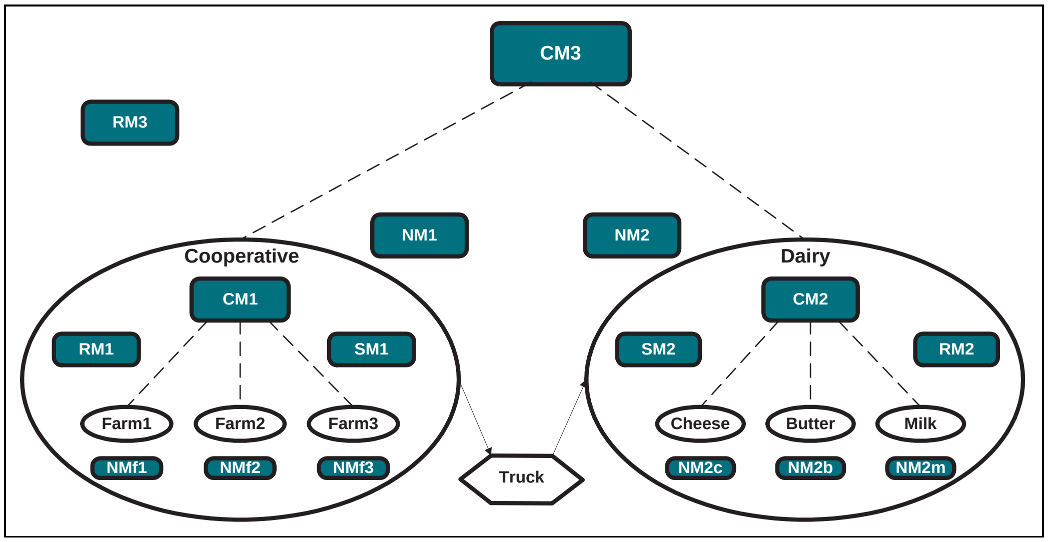


Figura 3.3: *Illustrating scope and manager hierarchies*

3.4.3 Revisiting the Case Study Using the Architecture

This section illustrates how chain dynamics are supported within the architecture. It starts by discussing coordination aspects, followed by negotiation aspects.

Coordination.

The first step in chain execution is its instantiation – this means that a plan's components are instantiated – e.g., Farm 3, registered in the Participant Repository, is a specific farm in Fig. 3.3. Farms 1, 2 and 3 are furthermore production elements. Each farm has its own negotiation manager (NMf1, NMf2, NMf3). Once the elements start being instantiated, they can agree to establish collaboration, according to coordination plans written and ran by a coordination manager (e.g., CM3). This begins chain execution, started by some coordination manager "higher-up" in the manager hierarchy (CM3) or by actor intervention.

The cooperative and the dairy may undergo several negotiation processes. Those negotiation processes are led by negotiation managers NM1 (for the cooperative) and NM2 (for the dairy). Negotiation is triggered by CM3. In this example, the cooperative and the dairy have their own regulation managers, namely RM1 and RM2. RM3 is responsible for handling regulations within the scope of CM3, and external to the scope of the dairy and cooperative.

Suppose now that CM3, as part of its plan, asks the cooperative to supply 5000 liters of milk the next day. None of the farms can singly afford that volume. Thus, CM1 coordinates this production. It may demand 1000 liters of one farm; 1500 liters of another; and 2500 liters of the last one. As soon as a farm gets the request ready, it reports to CM1. When all farms have reported, CM1 reports to CM3. This kind of communication and execution protocol is similar to that found in management of nested complex transactions in distributed systems [70].

Now, CM3 will ask some transportation agent (Truck) to collect the milk at the cooperative and deliver it to the dairy. When the milk arrives, Truck will notify CM3. CM3 will then ask the dairy to produce 100 liters of bottled milk, 50Kg of butter and 200Kg of cheese. CM2 takes care of this assignment, by coordinating the activities of butter, cheese and bottled milk units. Each unit reports the completion of its task to CM2. When all units have accomplished their tasks, CM2 reports to CM3, and so on.

In this scenario, CM1 and CM2 are subordinated to CM3, but they can coordinate plans that do not depend on CM3, for instance, related to their internal activities.

Negotiation.

The relationships among cooperative, farms, dairy (and the respective production units) is governed by contracts. The establishment of a contract is started by a coordination manager that requests intervention from negotiation managers. Consider, again, that CM3 asks the cooperative for a daily production of 5000 liters of milk for the next three months to be delivered to the dairy, but there is not any predefined quota for each farm. The negotiation happens at two distinct levels: the cooperative negotiates with the dairy through NM1 and NM2; the farms negotiate among themselves through NMf1, NMf2 and NMf3.

The negotiation sequence covering both negotiation levels is depicted in Fig. 3.4. First, CM3 asks the cooperative to deploy a contract negotiation with the dairy. This figure shows that, as soon as CM1 receives a negotiation request from CM3 (edge 1), CM1 starts two activities: a) It asks NM1 (edge 2) to negotiate the contract with the dairy´s NM (NM2); b) It asks (edge 3) Farm 1´s CM (CMf1) to start milk quotas negotiation among the farms.

As a consequence of CM1´s request, NM1 and NM2 develop a negotiation process. NM1 proposes contract clauses to NM2 (edge 4). The latter considers each clause individually and may accept it, reject it or propose an alternative (edge 5). The cycle proposal X alternative runs until they agree to or reject the clause. Eventually, NM1 and NM2 agree to the contract. At the same time, CMf1 asks NMf1 (edge 6) to begin quota negotiation with NMf2 and NMf3 (edges 7 and 8, 9 and 10).

When quota negotiation is finished, NMf1 reports this to CMf1 (edge 11), which in turn relays this information to CM1 (edge 12). Eventually, NM1 and NM2 agree on the deployment contract and NM1 reports the agreement to CM1 (edge 13). As soon as both negotiation processes (milk quotas and deployment contract) are finished, CM1 reports to CM3 (edge 14). Note that eventually NM1 might ask the Cooperative or NM2 might ask the Dairy about some negotiation parameters during a negotiation process. This kind of request is not depicted in this figure.

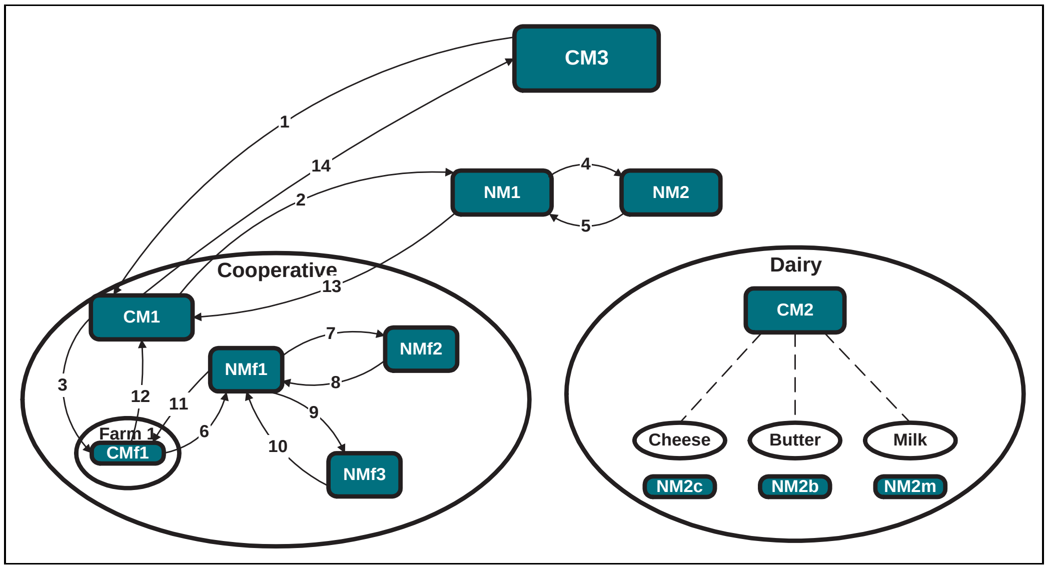


Figura 3.4: *Coordination and negotiation relationship*

A contract is executed and renegotiated on a clause-by-clause basis by the initiative of a coordination manager. For instance, the supply chain may have to be dynamically reconfigured due to a new factor (e.g. a new law, some natural disaster or animal epidemics in a region). Considering the managers illustrated in the Fig. 3.4, CM3 asks CM1 and CM2 to negotiate new parameters via the suitable negotiation managers NM1 and NM2. These, in turn, verify their contracts in order to determine which contracts and which clauses were affected. The affected clauses are renegotiated individually, again under the proposal X alternative cycle. Negotiation and renegotiation may need human intervention. Each new contract is stored in a Contract Repository by some negotiation manager.

Documentation.

The chain execution is documented into summaries that follow products along the chain. Summaries are in fact composed of sequences of local process and product summaries. They are updated at each chain step, and can be merged or subdivided.

Documentation proceeds along the chain. For instance, when the butter unit starts, a new process summary is created for its production process. At the end of this process, the butter unit's CM asks its summary manager to create a summary for the butter produced. This new butter summary is composed of a description of the butter fabrication process, appended to the input milk summary. Eventually, the dairy will output the butter to the next chain step, and this butter will be accompanied by its summary.

Regulation Upholding.

Coordination and negotiation involve regulation checking and upholding. For instance, at the end of butter production, CM2 may ask its regulation manager RM2 to check if the product satisfies the suitable restrictions. In order to do this, it will inform RM2 which constraints must be checked. Next, RM2 will combine information from Participant and Product repositories, plus data from the product summary to check these regulations, and return a verdict on regulation compliance, which is also stored in the summary.

3.5 Implementation

3.5.1 Mapping into Classes

Implementation of our architecture can be specified in terms of classes in an object-oriented system. Figure 3.5 uses UML and shows a high level specification of some of the topmost classes needed. The basic elements are in grey, managers are in black, repositories are in white. It shows that basic components include Storage, Transportation and Production, and also the possible compositions among them (Actors are not shown). Note the closed arrowheads from Production, Transport and Storage to Element. This indicates that Element generalizes the other classes, whereas black diamonds indicate composition – e.g., a Production element can encapsulate any other element, whereas Transportation and Storage elements cannot contain Production components.

Black arrows indicate responsibility relationships – e.g., a NM handles contracts, or a SM handles summaries. These classes are implemented in Java. The next section presents highlights of these classes.

3.5.2 Class Specification

CoordinationManager Class.

This class implements the CM block of Fig. 3.5. A Coordination Manager executes coordination plans. A coordination plan is composed by a set of activities. The coordination plan is a XML file that can be mapped to a BPEL4WS script. The values transferred to and from activities are also XML files. Each activity has an identification and may yield a result after completion. These 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 synchronization primitives.

A given plan can have more than one instance executing at the same time. Thus each plan execution has a unique instance identification. Each plan execution may also receive

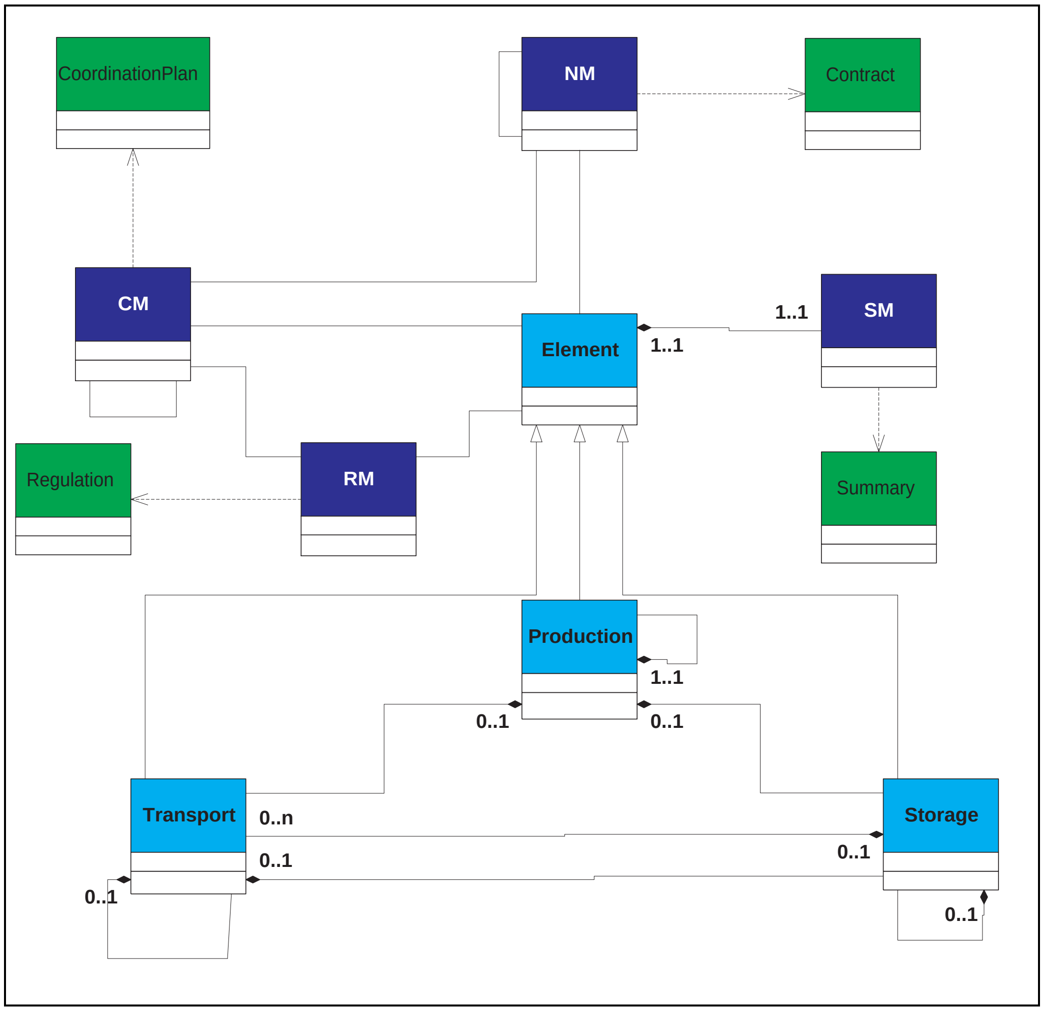


Figura 3.5: *Class diagram with emphasis in model components and management*

parameters from the environment.

A CM communicates with a CM within its scope (e.g., CM3 and CM1) via interfaces *CoordinationIF* (Fig. 3.6) and *ActivityReportIF* (Fig. 3.7). Orchestration is performed through these interfaces. The lower level CM receives the request through its *CoordinationIF* interface and reports the result to the parent's *ActivityReportIF* interface.

```
public interface CoordinationIF {

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

Figura 3.6: *CoordinationIF interface*

Figure 3.6 shows that the request for plan execution contains parameter *planAddr* that informs the address of the repository where the demanded plan is stored, *planId* is a key that identifies the plan inside the repository, *pars* are environmental parameters, *caller* is the address of the higher Coordination Manager, and *activityId* keeps both the activity and the instance identification of the higher activity that demanded the plan execution. The parameters *caller* and *activityId* are used to report the execution status to the higher manager.

Eventually, the lower manager reports the execution status to the parent manager. Using the received *caller* parameter, it can reach the higher manager and execute *reportPlanStatus* operation of the higher manager (Fig. 3.7). The parameter *st* informs the status (DONE, ACTIVE, SUSPENDED, RESUMED, CANCELED) and may convey some value produced by the plan´s execution, and *activityId* received previously is assigned to *id*.

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

Figura 3.7: *ActivityReportIF interface*

The Coordination Manager has another interface called *OwnerComponentIF* that is quite similar to *CoordinationIF* interface. This new interface is used by a component to demand an inner Coordination Manager the execution of a plan. The execution may be synchronous or asynchronous, and there is an operation to ask the status of an asynchronous plan execution.

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

RegulationManager Class.

This class implements the RM block of Fig. 3.5. An instance of this class verifies regulations. A regulation is evaluated against a summary of a product to verify if that product satisfies the constraints expressed in the regulation.

A Regulation is specified in an XML file (Fig. 3.8). It contains a section (tag *verify*) with the conditions that must hold for the regulation to be satisfied (the regulation is said to be satisfied). The evaluation of this condition may produce a certificate stamp - another XML file.

```
<regulation id=‘‘unique_id’’ type=‘‘CategoryName’’>
      <parameters> ... < /parameters>
      <enforce>
         <reg var=‘‘VarName’’ id=‘‘RegulationIdentification’’
            address=‘‘RegulationRepositoryAddress’’ >
            <par name=‘‘Parameter1Name’’> Parameter1Value< /par>
         < /reg>
      < /enforce>
      <verify> < /verify>
      <action>
         <ifok>
            <mark m=‘‘all|alltrue|allfalse|#VarName|##’’/ >
         < /ifok>
     < /action>
< /regulation>
```

Figura 3.8: *Regulation XML file*

Complex Regulations embed other regulations to be verified (tag *enforce*). The value produced by the evaluation of an enforced regulation is assigned to *VarName*. A complex regulation is satisfied iff its condition holds and so do all the enforced regulations.

The *action* tag indicates whether to store the certification stamps in the summary or not. The *mark* tag will instruct which mark is appended to the summary; e.g, *all* means that all stamps will be appended; *alltrue* appends the stamps whose value is yes; *allfalse* is the opposite; *#VarName*, appends the stamp contained in variable *VarName*; *##*, appends only the stamp produced by the composite regulation.

3.5.3 Implementation as Web Services

All architecture elements can be seen as implemented through or encapsulated by Web Services. The only exception is the coordination plan, which is mapped to a workflow.

In more detail, repositories and contracts are static entities encapsulated by Services that provide access to them. Actors can be either Services (e.g., a broker) or Service clients. All managers correspond to services, and the remaining architecture elements – Production, Transportation and Storage – are atomic services or the result of service composition via coordination plans.

The workflow that describes a coordination plan is constructed just as any workflow described in the literature [41], i.e.:

- totally predefined before execution; or
- constructed in an *ad hoc* manner by the CM responsible for the orchestration, while the chain is executed, typical of scientific workflows (e.g., [87, 30]); or
- a combination of both.

Each workflow activity references a service responsible for its execution. For instance, in figure 3.3, a coordination plan executed by CM2 is a workflow that contains an activity that starts cheese production. This activity must refer to the cheese unit (a Service), the desired kind of cheese (a Service for a Product Repository) and the regulations (a Service to a Regulation Repository) that must be verified during the production process in order to ensure cheese quality.

There follows the ennumeration of the interfaces of these Services, which can also be depicted as WSDL specification. Most of these blocks also implement an administration interface, used to configure the corresponding Web Service. The main interfaces of Transportation, Production and Storage Services are:

- *Interfaces for specific/business services:* each element represents a chain partner (e.g., business, enterprise, industry), and therefore can have one or more interfaces for its specific services.
- *Contract Negotiation interface:* receives requests from the Negotiation Manager about negotiation and contract parameters.
- *Contract Execution interface:* accepts requests from other components (or Coordination Manager) to execute a specific contract clause.
- *Sumary Management interface:* responsible for exchange and certification of summaries, via communication with the Summary Manager.

A Coordination Manager Service implements at least the following interfaces. The Java specifications of some of them are shown in section 3.5.2:

- *Coordination interface:* receives requests from a higher Coordination Managers. Orchestration happens through this interface.
- *Activity Report interface:* receives status reports about the activities demanded from another Service.
- *Owner Component interface:* the interface by which a Coordination Manager receives requests from the component that owns it.

The interfaces implemented by a Negotiation Manager Service include:

- *Negotiation Coordination interface:* accepts requests from the Coordination Manager.
- *Peer Negotiation interface:* for negotiation with another Negotiation Manager Service.

A Summary Manager Service has one *Exchange interface* for exchange of summaries among summary managers.

Finally, a Regulation Manager Service has one interface *Regulation Verifying interface.* It is responsible for checking all rules within a regulation against the chain's state. This may require requesting information from all repositories. It may be invoked by one or more chain components. The component that invoked it is responsible for enforcing the corresponding regulation.

All repositories are encapsulated by Services. The interfaces of these Services offer access to these data for retrieval and update. These interfaces can be accessed by the Managers of a chain and also by external services and systems that have no connection with a chain, but want to perform queries on products, participants, contracts and plans.

3.6 Related Work

There are several issues that can be analyzed under the umbrella of supply chains - e.g., concerning algorithms adopted, logistics, placement strategies, partner choice. One particular trend, called by [65] IT-related supply chains, concerns information technology tools and techniques to specify and implement such chains. In particular, a recent direction concerns the communication technologies adopted. Problems encountered in electronic commerce and B2B applications and interactions are the same as those faced by supply chain interactions [62].

Though there are many proposals for combining workflows and Web Services (e.g., [39] on agriculture) proposals for supply chains combining these mechanisms are still preliminary. The closest is the research on e-business using Web services, but for other goals – e.g., see [80]. [9] even states that the main reason for the lack of practical implementation of strategic supply chain development can be found in the high degree of complexity that is connected with the identification of supply chain entities and the modelling of the chain structure, as well as the high coordination effort.

Our goal is to contribute to solving these issues. Most researchers do not examine the entire chain, focusing only on some aspects. Auditing structures and log maintenance are ignored. Agricultural chains are mostly examined under a business or logistics framework.

Examples of such approaches are the work of [74] or [82]. The first 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 firm specifies the terms of trade across the entire supply chain and the coordinated behaviour is based on specification contracts. [82] discusses the usage of information technology in the american cattle-beef supply chain. The paper emphasizes the need for better information integration and well-defined means for describing and enforcing activitities coordination, negotiation and execution of contracts.

Since our proposal is based on Web services implementation, we also examine a few related issues. Two aspects have to be considered: mapping a chain's components to Web services and composition of these services.

[73] analyzes issues in service composition and comments on various standards for orchestration and choreography, such as BPEL4WS, WSCI and BPML. Important concerns in service execution in this context are long running transactions and exception handling. The actions in those standards are undone by compensation actions. This affects documentation of chain execution, since all performed actions are logged in summaries and in repositories. [25], in turn, overviews several proto-patterns for architecting and managing composite Web services, while [27] is more concerned with service semantics.

[24] proposes 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 [14] uses statecharts for defining service composition, and is based on a distributed orchestration engine.

[71] proposes a service-oriented architecture built upon the 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.

Several other authors are concerned with organizational and modeling aspects of supply chains, as indicated by the classification proposed by [65] to analyze efforts in supply chain modeling. This includes for instance work on partner coordination [59], logistics [84] or business contract languages [86].

3.7 Conclusions

This paper presented a framework for modeling, supervising and coordinating processes in agricultural supply chains. This framework is comprised of two parts: (i) a model for these production chains, that covers both declarative and dynamic aspects; and (ii) an architecture to support the model, based on Web Services and their interfaces.

The model takes into account the fact that agricultural chains are inherently heterogeneous, and sensitive to different kinds of constraints. Chain definition using this model involves specifying its basic components (Actors, Transportation, Process and Storage) and the components needed for cooperation, collaboration, negotiation and documentation (Contracts, Coordination plans, Regulations and Summaries). The model provides rules for composition and construction of these elements, thereby allowing *ad hoc* chain construction and execution. The model is mapped into an architecture of Web Services that provides support for contract negotiation, plan coordination, regulation enforcement and summary management. These services also encapsulate access to distinct repositories, that contain data on the chain's partners, processes, regulations, constraints, contracts and execution documentation. This architecture supports flow execution at two dimensions: within and across enterprises, for a multiple hierarchy of coordination levels, under service orchestration. Service coordination encompasses global and local dynamics, enforceable by communication protocols established among and across coordination levels.

The main contributions are thus the following: (1) an information technology-based model for specification of agricultural supply chains, which takes into consideration scope, structure and goals, and supports coordination, cooperation and documentation; (2) an architecture for its implementation, which emphasizes negotiation, regulation management, coordination and service flow issues; (3) validation of the model via a real life case study in agriculture.

Current work includes refining the object model of the framework, which will in turn allow implementation and testing of the architecture. This includes testing the suitability of scientific workflows to support the dynamics of ad-hoc coordination plan construction. The implementation will be tested against case studies provided by Brazil's agriculture ministry research corporation.

Capítulo 4

A Negotiation Process and a Contract Format for Agricultural Supply Chains

E. Bacarin, E.R.M. Madeira and C.B. Medeiros. Contract E-Negotiation in Agricultural Supply Chains. *Intl. Journal of Electronic Commerce*, 12(4): 71–97, summer, 2008.

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.

4.1 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 [65]. 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 [12].

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 4.4, (iii) – Section 4.3 and (iv) – Section. 4.5.

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 4.2 describes our model for agricultural supply chains and an architecture induced by this model. Section 4.3 describes the organization of our e-contracts and Section 4.4 describes the e-negotiation process to produce such a contract. Section 4.5 discusses some implementation issues and a few basic interactions scenarios, such as ballots, auctions, and quota negotiation. Section 4.6 discusses related work, mainly on contracts and negotiation. Finally, Section 4.7 concludes the paper.

4.2 The Model and Basic Architecture

Our framework is based on a specific model for supply chains (see [21]). 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 4.1 illustrates the main concepts of our model, showing a simplified chain for the motivating example (Section 4.1). 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 [57]. 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 elements, 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 di-

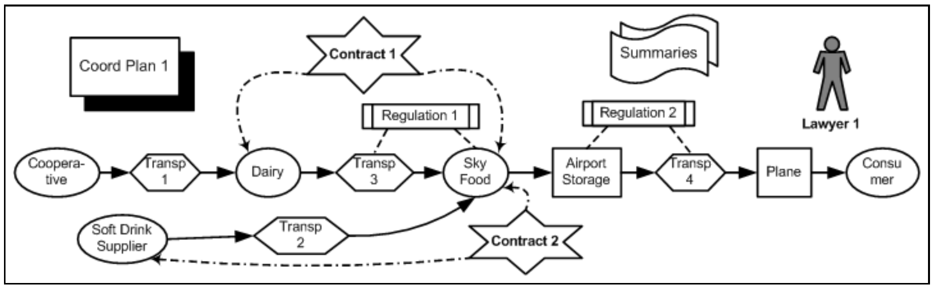


Figura 4.1: *Simplified supply chain*

rectly 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 [21].

4.3 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 4.3.1 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 4.4 filling the blanks. If the negotiation succeeds, an actual contract instance is produced – Section 4.3.2. 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.

4.3.1 The Contract Model

Our contract model is a template written in XML. Figure 4.2 shows a graphical representation of its structure and Figure 4.3 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 4.2): *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 4.2) 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.

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.

Figure 4.3 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

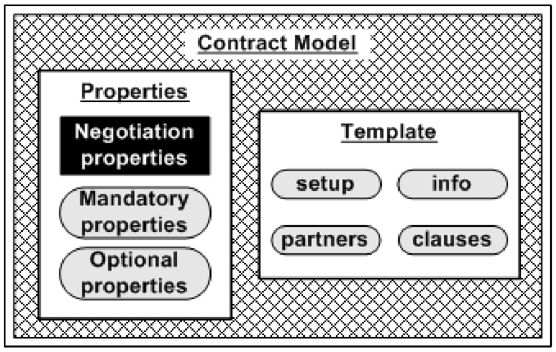


Figura 4.2: *Contract model schematic representation*

```
<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>
```

Figura 4.3: *An Excerpt of a Contract Model, Focusing Properties*

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.

4.3.2 Contract Instance

A contract is based on a contract model (Section 4.3.1). 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 4.4) 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 4.2). 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>
```

Figura 4.4: *Contract Partners*

The properties section. This section is directly derived from the property declaration in the contract model (Section 4.3.1). Properties are value containers (like variables or constants). Their values may be fixed (static) or defined during contract enactment (dynamic). Figure 4.5 presents the property section derived from the contract model of Figure 4.3. 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.

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 4.6) is identified by

```
<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>
```

Figura 4.5: *Examples of Properties*

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 4.6 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 4.6). 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>
```

Figura 4.6: *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 4.1 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.

Tabela 4.1: Examples of *mode* combination for Figure 4.6

4.4 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 4.4.1 gives an overview of how negotiations are set up; the subsequent sections detail the negotiation process itself.

4.4.1 Organization of the Negotiation

Negotiators are instances of the Negotiation Managers (NM) of our architecture (Section 4.2). 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 [42].

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 4.4.2): 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.

4.4.2 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 4.5.2. 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 4.5.2. 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.

4.4.3 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 4.7 presents an excerpt of the grammar. Non-terminals are capitalized, terminals are in bold, ◇ means another rule for the same non-terminal, [A] means that A is optional, A+ means one or more A.

```
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]
```

Figura 4.7: *Excerpt of grammar for property negotiation*

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., 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 4.8 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 4.9 shows a short bargaining process, where N2 agrees to N1's counter-offer.

```
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□
```

Figura 4.8: *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□
```

Figura 4.9: *Example of bargaining given an RFP*

Figure 4.10 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 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 4.11 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.

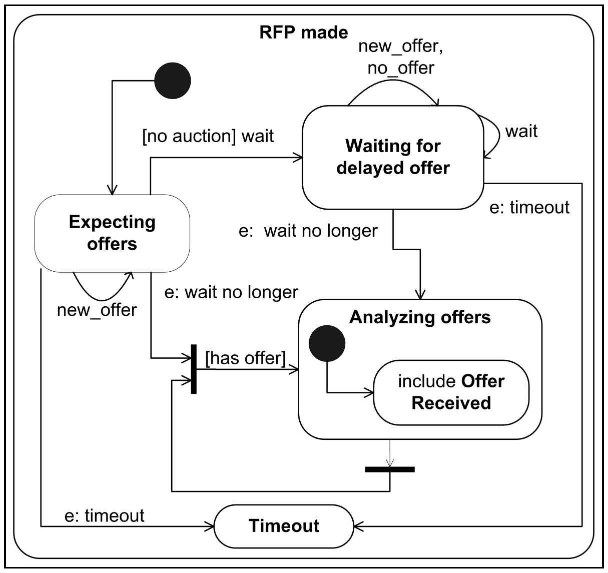


Figura 4.10: *Leader's state after issuing an RFP*

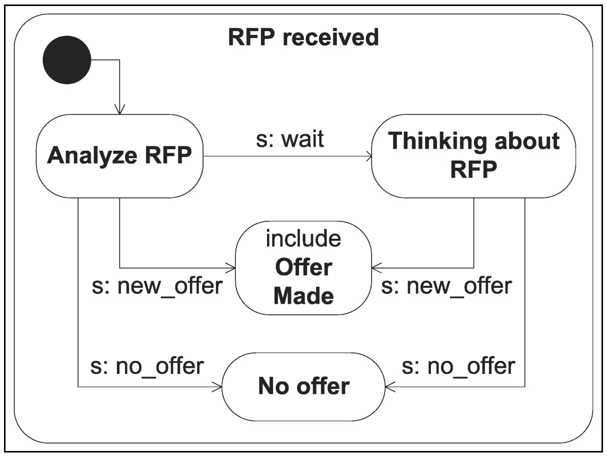


Figura 4.11: *Negotiator's state after receiving an RFP*

Note that the communication primitives presented in Figure 4.7 allow several 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 4.5.2, 4.5.2, and 4.5.2 show a few examples of different negotiation styles.

4.5 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.

4.5.1 Interfaces

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 [53].

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.

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.

4.5.2 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 4.12 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 4.2 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)

Tabela 4.2: Interface acronyms

Auctions

Recall Figure 4.1. 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 4.12 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.

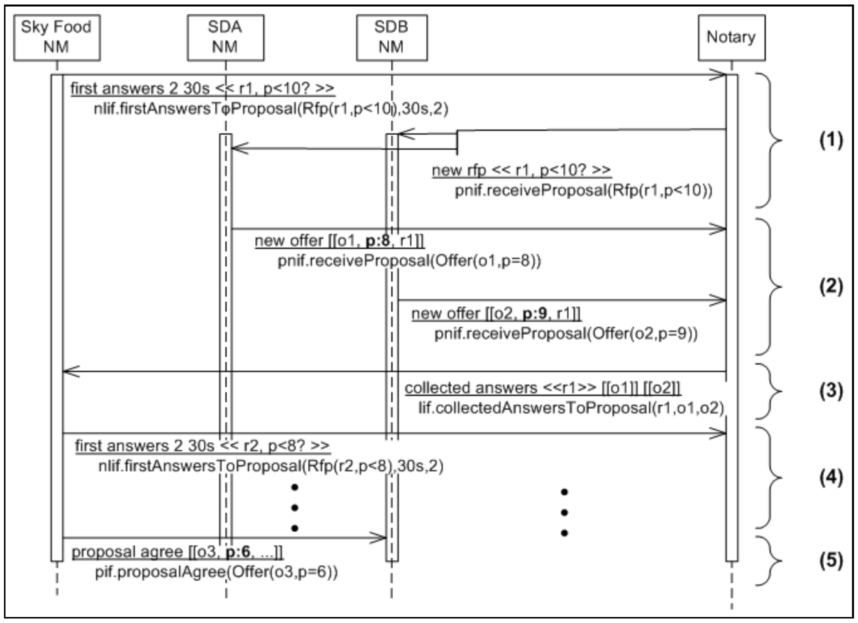

Sky Food NM
SDA NM
SDB NM
Notary
first answers 2 30s << r1, p<10? >>
nlif.firstAnswersToProposal(Rfp(r1,p<10),30s,2)
new rfp << r1, p<10? >>
pnif.receiveProposal(Rfp(r1,p<10))
new offer [[o1, p:8, r1]]
pnif.receiveProposal(Offer(o1,p=8))
new offer [[o2, p:9, r1]]
pnif.receiveProposal(Offer(o2,p=9))
collected answers <<r1>> [[o1]] [[o2]]
lif.collectedAnswersToProposal(r1,o1,o2)
first answers 2 30s << r2, p<8? >>
nlif.firstAnswersToProposal(Rfp(r2,p<8),30s,2)
proposal agree [[o3, p:6, ...]]
pif.proposalAgree(Offer(o3,p=6))
(1)
(2)
(3)
(4)
(5)

Figura 4.12: *The english 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.

Voting

Recall again Figure 4.1. 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 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 4.13 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.

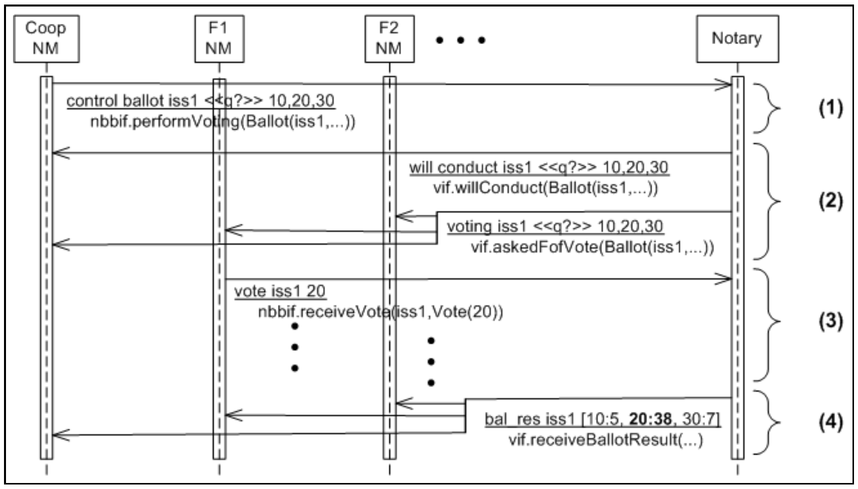

Coop NM
F1 NM
F2 NM
Notary
control ballot iss1 <<q?>> 10,20,30
nbbif.performVoting(Ballot(iss1,...))
(1)
will conduct iss1 <<q?>> 10,20,30
vif.willConduct(Ballot(iss1,...))
voting iss1 <<q?>> 10,20,30
vif.askedFofVote(Ballot(iss1,...))
(2)
vote iss1 20
nbbif.receiveVote(iss1,Vote(20))
(3)
bal_res iss1 [10:5, 20:38, 30:7]
vif.receiveBallotResult(...)
(4)

Figura 4.13: *Voting for maximum quota per farm*

Quota Negotiation

This section resumes the scenario presented in Section 4.5.2, 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.
- 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 4.3): for a given RFP, the 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.

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

Tabela 4.3: The price function for a farm

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:

$$C_T = Q_1 * P_1 + Q_2 * P_2 + ... + Q_k * P_k \tag{4.1}$$

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 4.1, 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 4.14 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.

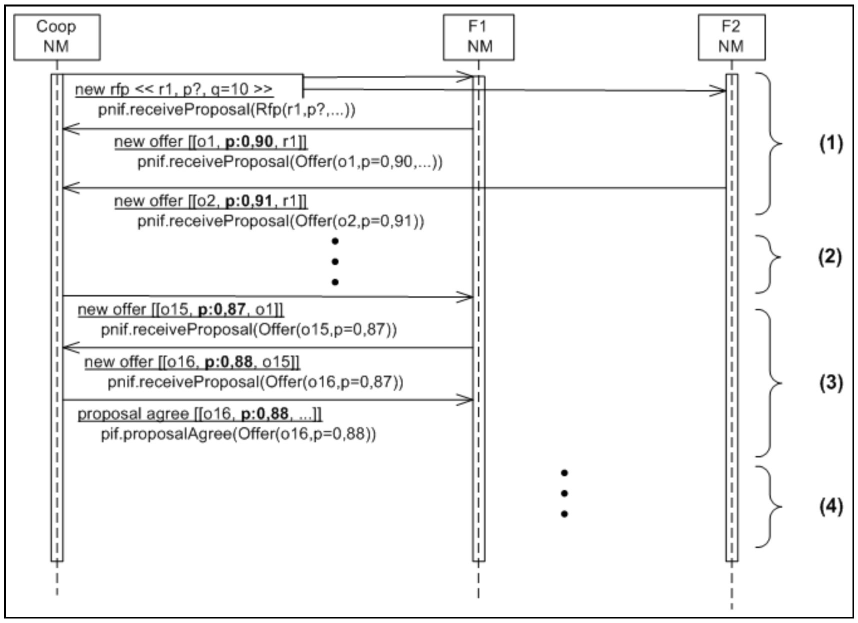

Coop NM
F1 NM
F2 NM
new rfp << r1, p?, q=10 >>
pnif.receiveProposal(Rfp(r1,p?,...))
new offer [[o1, p:0,90, r1]]
pnif.receiveProposal(Offer(o1,p=0,90,...))
new offer [[o2, p:0,91, r1]]
pnif.receiveProposal(Offer(o2,p=0,91))
new offer [[o15, p:0,87, o1]]
pnif.receiveProposal(Offer(o15,p=0,87))
new offer [[o16, p:0,88, o15]]
pnif.receiveProposal(Offer(o16,p=0,87))
proposal agree [[o16, p:0,88, ...]]
pif.proposalAgree(Offer(o16,p=0,88))
(1)
(2)
(3)
(4)

Figura 4.14: *Quota negotiation*

4.6 Related Work

This section reviews related work on supply chains (Section 4.6.1), electronic contracts (Section 4.6.2), the negotiation process (Section 4.6.3), and contract enactment (Section 4.6.4).

4.6.1 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 [48]. It aims at specific processes in order to optimize inventory level, reduce cost and increase profits. The proposal of [58] 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 [31] 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.

4.6.2 Contracts

There are several proposals for e-contract specification. In [11], 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.

[60] 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 [58], the entities of an e-contract are: parties, clauses, budget, roles and payments. Contracts can be composed into more complex ones. [51] 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 [86], 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 [21], 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 [37] 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.

4.6.3 The Negotiation Process

There are a number of mechanisms that guide the negotiation process. Bartolini [22] constructs negotiation templates that specify different negotiation parameters that can be constrained or open. Chiu [33] 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, [77] uses a contract template that describes the negotiation parameters, how they are interrelated, along with meta-level rules about the negotiation. In contrast, [50] 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., [51] or [68] – do not present a proper negotiation process. Their approach is based on matchmaking. The approach of [51] 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. [68] 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 [22], 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 [43], 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 [43] and Courteous Logic Programming [46]. 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 [61], 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, [61] 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 [40], e.g., [61].

Governatori and others [43] 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 4.4 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

Tabela 4.4: Negotiation issues

4.6.4 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 [60] expresses and monitors conditions in business contracts. In addition, [60] 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 [8, 4]. Governatori and others [44] have a different approach. They aim at checking the compatibility of business processes and business contracts by means of a logic-based formalism.

4.7 Conclusions

The paper presented a framework for contract negotiation in agricultural supply chains, which uses our supply chain model ([21]). 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 4.2) – 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 negotiation 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.

Capítulo 5

Assembling VOs

E. Bacarin, E.R.M. Madeira and C.B. Medeiros. Assembling and Managing Virtual Organizations out of Multi-party Contracts. In *ICEIS* 758-769, 2009.

Abstract
Assembling virtual organizations is a complex process, which can be modeled and managed by means of a multi-party contract. Such a contract must encompass seeking consensus among parties in some issues, while simultaneously allowing for competition in others. Present solutions in contract negotiation are not satisfactory because they do not accommodate such a variety of needs and negotiation protocols. This paper shows our solution to this problem, discussing how our SPICA negotiation protocol can be used to build up virtual organizations. It assesses the effectiveness of our approach and discusses the protocol's implementation. **Key words:** virtual organization, multi-party contract, supply chain, negotiation, auction, ballot, bargain.

5.1 Introduction

Virtual Organizations (VO) are dynamic alliances of enterprises that together can take advantage of economies of scale when available [83]. Assembling and managing them is a complex task, due to the many relationships and agreements among their components. One possible way to shape and manage such organizations is via multi-party contracts, which must reflect obligations, rights and interaction modes within a virtual collaboration scenario. They are built by means of some negotiation mechanism.

According to [36], VOs need negotiation protocols that are multi-party and interactive, i.e., the protocol should allow simultaneous negotiation among three or more partners and they should be allowed to refine a received proposal, e.g., by means of a counter-proposal.

The process of constructing a VO can be quite complex: the partners should reach a consensus on some issues, whereas there is competition among them on others, and also other issues may demand individual agreement. While existing solutions do not make allowance for these negotiation heterogeneity, our protocol provides mechanisms to develop all those negotiation styles for negotiating a single contract, namely, ballots, auctions and bargains.

The same mechanisms can also be employed individually to build specific marketplaces. For instance, they can be easily configured to provide different auction styles (e.g., English and Dutch auctions). This paper highlights such mechanisms by means of showing how they can be set up for a number of distinct marketplaces.

The main contributions of the paper are: (a) it points out that multi-party contracts are a means for describing virtual organizationss; (b) it shows how to integrate three different styles of negotiation (bargain, ballot, auction) to build up a virtual organization using SPICA negotiation protocol; (c) it presents some details of the implementation of the SPICA negotiation protocol.

The paper is organized as follows. Section 5.2 presents a running example that will be used throughout the paper. Section 5.3 shows how SPICA negotiation protocol can be used to build different setups of marketplaces. Next, Section 5.4 describes briefly the protocol's implementation. Then, Section 5.5 evaluates the approach proposed in the paper. Section 5.6 presents related work. Finally, Section 5.7 concludes the paper.

5.2 The Running Example

The scenario described in this section is used throughout the paper to motivate and exemplify the usage of the SPICA negotiation protocol.

The scenario consists of a number of farms (F_1,F_2,...,F_n), a few orange processing companies (PC_1,PC_2,...,PC_i) and a railway company (RC). The farms grow orange trees and deliver their crops to the processing companies. A processing company produces concentrated orange juice to be exported. The juice is transported from the company's plant to the nearest harbour by means of the only railway company available in that region.

There is a standard contract model that processing companies use to buy oranges. The main contract's provisions are shown in Figure 5.1.[1] Note that *pj*, *ff* and *pf* are the model's parameters.

The farms have organized themselves into a Cooperative to better negotiate delivery contracts. The cooperative will choose a processing company and will establish a delivery

[1]In fact, these provisions are a simplification of a contract model (*Consecitrus*) Brazilian farmers and orange juice industry have been discussing to be used in negotiation of future crops.

(1) The price paid by the PC will cover the production costs plus a percentage of the juice value (*pj*) in the commodities market.
(2) The harvest and transportation from farm to industry is done at the PC's expense.
(3) If the supplier is farther than 100km from the processing company, the farm will pay an extra freight fee (*ff*).
(4) If the farm's productivity is below a certain local level, the farm must also pay an extra fee (*pf*).

Figura 5.1: *Contract model for orange crop.*

contract. The chosen company will be the one that proposes the best values for parameters *pj*, *ff* and *pf*. Thus, the farms must reach a consensus on what is the best proposal. Finally, the processing company will negotiate the juice transportation with the railway company. They will haggle over the freight fee *rff*.

Notice the following peculiarities in this scenario. There are several competing PCs and the cooperative will choose the best one using an *auction*. However, the farms must agree on this choice via a *ballot*. Since there is just one railway company, the PC is forced to *bargain*. This complex scenario requires a contract that contemplates multi-party negotiation, and bargains, ballots and auctions. As will be seen, we provide a seamless solution that supports these requirements.

5.3 The SPICA Negotiation Protocol

The negotiation process is guided by a contract template. Negotiators exchange messages that comply with the SPICA negotiation protocol. If there is an agreement, a contract instance is produced. Section 5.3.1 describes the contract template and the contract instance. Section 5.3.2 describes the protocol.

5.3.1 Contract Templates and Contracts

A contract template consists of a set of clauses with blanks to be filled in. Such blanks are referred to by so-called *properties* and the negotiation process aims at assigning values to them. Thus, a contract instance is a contract template with its properties successfully negotiated. The obligations (or rights) stated in a clause may bind (or benefit) several partners.

The contract model depicted in Figure 5.1 gives rise to a contract template for the scenario presented in Section 5.2. A contract template is an XML document composed of several sections. One of them contains a set of clauses, the others are used for setting up the negotiation environment. A template's clause respective to the model is presented in Figure 5.2. It is written in plain English for simplicity.

```
text:  The price paid by the @OBLIGED to the
@AUTHORIZED will cover the production costs plus
a percentage worth of #PJ of the juice value in
the commodities market.
depends:  (a precondition)
enforces:  (a set of regulations)
service:  (URLs and other parameters)
authorized:  F1,F2,F3,F4,...       obliged:  PC
```

Figura 5.2: *A simplified template's clause.*

Note that the name of the property to be negotiated (PJ in the figure) is embedded within the clause's *text*. There are a few parameters regarding the clause enactment. There is a precondition (*depends*) that must hold before the processing company should make the payment. In this example, it could be the existence of an formal statement of a trusted company about the juice's price in the market. There is a postcondition (*enforces*) that refers to one or more regulations that must hold after clause enactment. Regulations state a number of conditions to a product be transfered to a partner to another. In this example, it could be the accomplishment of a few legal procedures established by the local government; in other context, it could be a quality criterion to be meet. *Service* points to a business process to be executed, enacting the clause. *Authorized* lists the parties that will be payed and *obliged* lists the names of the processing companies that will pay the due value. These lists are referred to by the special properties @AUTORIZED and @OBLIGED in the *text*.

5.3.2 The Protocol

The main data exchanged in a negotiation by means of negotiation messages are *request for proposals* (RFPs), *offers*, *requests for information* (RFIs) and *information* (Info). They convey several parameters that tune up a specific negotiation, identify the sender and the receivers, and help establishing correlation among messages. Only the relevant parameters for the purpose of this paper are presented.

An RFP invites another party to negotiate a set of properties. A negotiator A sends an RFP to a negotiator B asking for a value for one or more properties. More specifically, an RFP conveys three pieces of data: a set of *asked properties*, a set of *assigned properties* and a *restriction*. The example below shows two RFPs. They are written using a simplified notation. The first RFP asks values for properties *ff* and *pp* and imposes a restriction over the value for *ff*: it would only accept values lesser than 3.00, but does not impose any restriction over *pp*. The second one proposes a value for *ff*, asks a value for *pf*, but imposes a restriction over *pf*. The symbols ≪ and ≫ enclose an RFP.

```
(1)    ≪----,{pp,ff},‘ff<3.00’≫
(2)    ≪{ff:1.96},{pf},‘pf<1.15’≫
```

A negotiator A proposes a value to one or more properties sending an offer to a negotiator B. Negotiator A informs the properties it is interested in and the values it proposes for them. If negotiator B accepts the offer, both negotiators are committed to the proposed values. A negotiator answers an RFP sending back an offer that assigns values to the asked properties. The example below shows offers that answers the previous RFPs. The first offer is a valid answer for RFP (1). It assigns the value 2.3 to property *pp* and the value 2.50 to property *ff*. This value complies with the imposed restriction. The second offer is a valid answer for RFP (2). The proposed value for property *ff* must be exactly the same of the one assigned in the respective RFP. The symbols [and] enclose an offer.

```
(1)    [pp=2.3, ff=2.50]
(2)    [pp=0.9, ff=1.96]
```

An RFI is very similar to an RFP: it asks values for properties, but also lower and upper bounds for them. An Info is similar to an offer: it proposes values for asked properties and also informs upper and lower bounds for them, however, the negotiator which issued an Info is not committed to it.

RFPs and offers are used to build several styles of negotiation that boil down to three basic ones: bargain, ballots, and auctions. Other styles are obtained from different setups of these basic ones. They use a few negotiation messages that are introduced by the examples that follow. RFIs and Infos do not lead to agreements (they are not committing), but they help constructing better proposals, thus improving the negotiation process.

The scenario presented in Section 5.2 uses all those styles. Firstly, only one processing company will be chosen. Thus, there is competition among them to decide which one will assign values for properties *pj*, *ff* and *pf*. This is resolved by means of an auction.

However, consensus is also needed: the farms must agree on a received bid. This is dealt through a ballot. Finally, the winning processing company has to negotiate property *rff* with the railway company. Since there is only one such a company, a bargain is used.

There are two approaches to organize the negotiation environment. The first one should consist of three separated marketplaces: one runs the auction and selects the winning bid. Then, this bid is submitted to a ballot, verifying if it is accepted by the most of the farms. Finally, the winning (and accepted) processing company will use a bargaining marketplace to negotiate the transportation. The second environment should consist of an integrated marketplace that develops all those negotiations.

Section 5.3.3 presents the first approach. It shows each negotiation style separately. Section 5.3.4 shows how the negotiation styles can be integrated in one marketplace.

5.3.3 Individual Marketplaces

English Auction

This first individual scenario shows the cooperative looking for a processing company which would buy the farms production in a specific season. Recall that the negotiation parameters (i.e., properties) are *pj*, *ff* and *pf* (Section 5.2). There are several candidate processing companies (PC1,PC2,...). The cooperative (Coop) will choose one of them by means of an auction. It is the auctioneer and it is helped by a notary (Not) which is a trusted third-party. Figure 5.3 shows the auction running. **(1)** The cooperative asks the notary to collect up to 2 bids within 30s. The auction's subject is described by an RFP. This RFP asks values for properties *pj*, *ff* and *pf*, and imposes a restriction over the proposals for *pj* — only values higher than 1.2 will be accepted. There should be restrictions over the other properties. They were omitted for the sake of simplicity. The notary agrees to run the asked auction step (it might have refused instead). **(2)** The notary broadcasts the auction issue to all bidders. **(3)** The bidders send offers back to the notary. The offers assign values to the asked properties. The notary collects the received bids (i.e., offers) and sends them to the cooperative. The offers are represented by *o1* and *o2* in the figure. **(4)** The cooperative analyses the received bids and comes to conclusion that it could get better bids. Thus, the cooperative builds a new RFP (note the new restriction over *pj*) and asks the notary to develop another auction step. This repeats as many times as wanted. Eventually, no bidder is interested in submitting another bid. Then, **(5)** the cooperative agrees on the best bid of the previous auction step. The fact that the demanded auction step asked for 2 bids does not imply that the auctioneer must agree on 2 bids.

A Dutch auction can be easily run using the same mechanism. However, the auction's subject is described by means of an offer. In step (1), the auctioneer sends an offer to

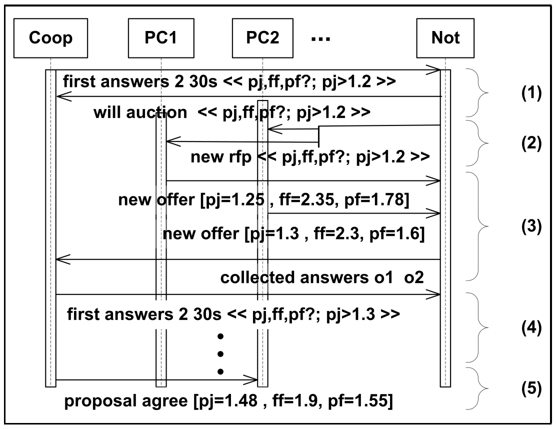

Coop
PC1
PC2
...
Not
first answers 2 30s << pj,ff,pf?; pj>1.2 >>
will auction << pj,ff,pf?; pj>1.2 >>
new rfp << pj,ff,pf?; pj>1.2 >>
new offer [pj=1.25 , ff=2.35, pf=1.78]
new offer [pj=1.3 , ff=2.3, pf=1.6]
collected answers o1 o2
first answers 2 30s << pj,ff,pf?; pj>1.3 >>
proposal agree [pj=1.48 , ff=1.9, pf=1.55]
(1)
(2)
(3)
(4)
(5)

Figura 5.3: *An auction.*

the notary. In (3), the bidders which first agree on that offer win the auction (up to 2 within 30s) . If no bidder agrees on such an offer within 30s, the auctioneer (4) will build another offer and submit it to another auction step. This repeats until there are winning bidders or the auctioneer gives up.

Ballot

The cooperative has found a processing company (the winning bidder in Section 5.3.3). However, such a choice must be validated by the farms. Now the cooperative runs a ballot helped by the notary (Figure 5.4). **(1)** The cooperative asks the notary to run a ballot. The ballot issue is described by means of an offer, i.e., the winning bid. The notary accepts conducting the ballot (it might have refused). **(2)** The notary broadcasts the issue to the voters (farms). The cooperative is not a voter in this case. It would be a voter in other setups. **(3)** The farms send their votes to the leader. In this case, they can only agree (vote *ok*) or disagree (vote *nok*). Abstention is also possible. **(4)** The notary collects the votes, counts them and broadcasts the result to all parties (farms and cooperative). In this example, the farms have accepted the winning bid, because more farms have agreed (15) than disagreed (8)

The presented scenario was a “take-it-or-leave-it” ballot. The voters could only have accepted or refused the proposed offer. In step (1), an RFP is used instead of an offer when there are several alternatives for the same property. In this case, the vote will contain one of these alternatives (instead of *ok* or *nok*). In some setups, a few voters may have veto power. If one of them sends a veto instead of a vote, the ballot is voted down.

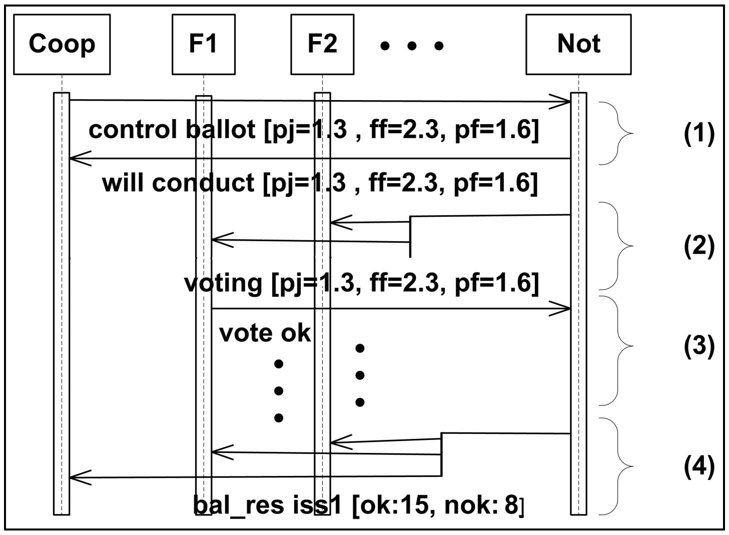

Coop
F1
F2
Not
control ballot [pj=1.3 , ff=2.3, pf=1.6]
will conduct [pj=1.3 , ff=2.3, pf=1.6]
voting [pj=1.3, ff=2.3, pf=1.6]
vote ok
bal_res iss1 [ok:15, nok: 8]
(1)
(2)
(3)
(4)

Figura 5.4: *A ballot.*

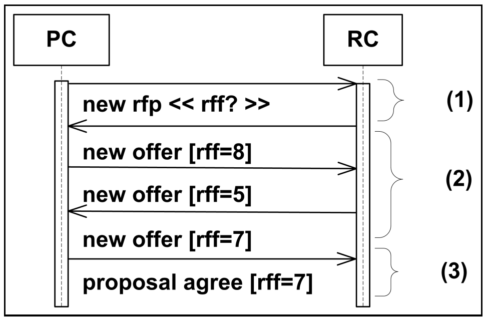


Figura 5.5: *A bargain.*

Bargain

The processing company (PC) has been chosen and validated by the farms. Now it has to negotiate the transportation with the railway company (RC). They will haggle over a value for property *rff*. This is shown in Figure 5.5. **(1)** PC asks the freight cost RC would charge for the transportation. **(2)** RC answers that it would charge 8. PC considers is too expensive and makes a counter-offer: 5. RC finds the proposal too cheap and makes another counter-offer: 7. This cycle of counter-offers is repeated as much as needed. **(3)** The process finishes when PC (or RC) reaches a final decision — in this case, PC agrees on the offer.

5.3.4 Putting Marketplaces together

The approach presented in Section 5.3.3 has two main drawbacks. Firstly, the auctioneer has to develop completely an auction and use its own criteria (not the farms') to choose the winning bid. Secondly, it has to submit its chosen bid to a ballot. If the bid is not approved, the auctioneer has to start the auction anew.

In an alternative setup, auction steps are interleaved with ballot sessions: the auctioneer submits all collected bids to the farms at the end of the auction step in successive ballots. If one of the bids is approved, the auctioneer chooses this as the winning bid; otherwise, it runs another auction step.

The role of the auctioneer in this setup can range from neutral to highly interested in a certain outcome. The order that it submits the received bids to ballot may influence the result. Thus, it can rank the bids aiming at efficiency (e.g., submitting first the bids it considers being more probable to be accepted) or according its own interests.

The voters determine the auction step's winning bid. In case no bid is chosen, the auctioneer typically will create a new RFP (or offer), and submit it to a new auction step. The auctioneer can consider the ballot's result of previous steps and try to prepare an RFP that would direct the bids closer the voter's expectations.

5.4 Implementation in Brief

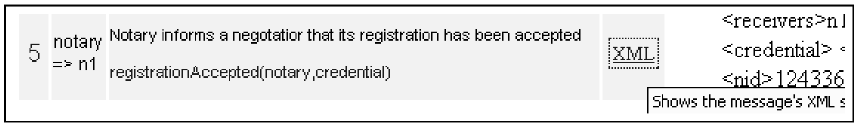
We have been implementing a framework for the integration of agricultural supply chain (SPICA). The negotiation of contracts is a part of it (SPICA Negotiation Protocol). The core of this negotiation protocol has been implemented. This section presents a few details about such an implementation.

Negotiators and the notary are web services. A negotiator N1 is willing to interact with another negotiator N2 (it might also be the notary). Such an interaction happens by means of an operation being invoked at the appropriate interface.

The framework provides a number of Java classes and Java interfaces to ease the implementation of negotiators. The Web services interfaces described in [16] are directly mapped to Java interfaces. A negotiator should react properly upon receiving a given negotiation message. For instance, whenever a negotiator receives an offer, it should (a) analyze it and decide about agreeing, disagreeing or making a counter-offer; (b) prepare the corresponding answer message, and (c) send it back to the offer originator. There is a default negotiator (DN) that implements this mechanism in a way that a specific negotiator (SN) needs only to override a few methods concerning the decision-making phase (i.e., step a).

There are also classes that help the communication of negotiation messages among negotiators. The framework's design aims at providing specific negotiators the illusion that they are exchanging negotiation messages by means of simple method calls at another local object (i.e., the other negotiator(s)). To do so, the framework provides two classes: *CommunicationAdaptor* and *MessageBroker*. *CommunicationAdaptor* mimics a negotiator. When a specific negotiator (SN1) wants to call a method M at another negotiator (SN2), it calls this method M of the *CommunicationAdaptor*. Then, the *CommunicationAdaptor* serializes the method's parameters (XML-formated) and delivers it to a middleware that transports the message. At the other end, the message reaches a *MessageBroker* which gets the message's parameters and invokes the method M at SN2.

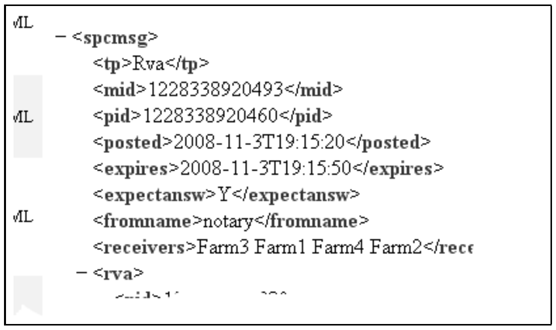
Figures 5.6 and 5.7 show an excerpt of messages exchanged among negotiators logged by the system. In Message 77 (Figure 5.6), the notary asks the negotiators (farms) to vote on a specific issue. The notary uses the *CommunicationAdaptor* to upload this message.



```
5   notary   Notary informs a negotiatior that its registration has been accepted      XML    <receivers>n1
    => n1                                                                                     <credential> <
             registrationAccepted(notary,credential)                                          <mid>124336
                                                                                       Shows the message's XML s
```

Figura 5.6: *Logged message.*

In the end, the method *askedForVote* is called at each negotiator. Figure 5.7 shows an excerpt of the respective message serialized by the *CommunicationAdaptor*.



```
ML
     - <spcmsg>
         <tp>Rva</tp>
         <mid>1228338920493</mid>
ML       <pid>1228338920460</pid>
         <posted>2008-11-3T19:15:20</posted>
         <expires>2008-11-3T19:15:50</expires>
         <expectansw>Y</expectansw>
ML       <fromname>notary</fromname>
         <receivers>Farm3 Farm1 Farm4 Farm2</rece
       - <rva>
```

Figura 5.7: *XML serialized message.*

5.5 Discussion

This paper shows by means of an example how a multi-party contract can be negotiated using the SPICA negotiation protocol.

Section 5.3 presented the SPICA protocol. The individual marketplaces approach (Sec. 5.3.3) was discussed first because most of the negotiation frameworks found in the literature have only one negotiation style, typically auctions or bargains. To the best of our knowledge, none uses a ballot as a foundation of a marketplace. If one would create a marketplace combining an existing auction and a bargain framework, it would be like the one proposed in Section 5.3.3.

Section 5.3.4 showed that the same primitives can be combined into an integrated marketplace. Thus, it is possible to explicitly correlate different negotiations. The contract instance produced by such a negotiation naturally shapes a VO. It is noteworthy that VOs are better shaped by means of multi-party contracts than by a set of bi-lateral ones. In this context, negotiation by consensus is quite important but rarely used.

A contract template can be used to describe a VO and its negotiation is the process of building a new VO that endures until the end of the agreed contract. For instance, Section 5.2 showed a scenario consisting of several actors. It gives rise to two possible approaches. In the first, two unrelated bi-lateral contracts are built: (a) between the processing companies and the cooperative, and (b) between the processing company and the railway. In this setup, the railway company is not aware about its role in the VO. In the second approach, a multi-party contract is used to establish the relationship among farms, processing companies and the railway company. The model presented in Figure 5.1 should only be enhanced with a clause about property *rff*. This setup allows that the contract provisions focus on a shared goal (i.e., export orange juice and improve profit), shaping a VO where all partners know their role in it.

The SPICA negotiation protocol was designed to provide flexible yet comprehensive negotiation primitives. They are somehow choreographic in the sense that they indicate how the partners should react upon receiving a given message, but does not fully define their expected behaviour. Auctions are a clear example of this. The protocol just defines that the auctioneer will ask the notary to conduct one auction step (not the whole auction). The notary will inform the negotiators about it, collect the bids and send them back to the auctioneer. The auctioneer is the one who is responsible for deciding if there is a winning bid, or if it will try another step, and, even, if it would give up the auction at all. This gives rise to several opportunities. For example, it is possible to run different styles of negotiation without changing the protocol at all. For instance, in Section 5.3.4 the winning bid (if any) of an auction step is decided by means of a ballot and not by the auctioneer itself. Another example, a segment of the supply chain or a VO may specify different types of behaviour of auctioneers and establish that negotiations within such segment or VO will be done under such specification.

The negotiation primitives build on two basic concepts RFPs and offers. RFIs and Infos may be used for building better proposals to be submitted to a ballot. For instance, a processing company could use RFIs to have a learned guess about the transportation costs and use this knowledge to submit more competitive bids. In this case, it would be a three-level negotiation scenario: (i) a processing company submits RFIs to the transportation company and (ii) take part of an auction, (iii) and the winning bid is decided by means of a ballot, all simultaneously.

5.6 Related Work

Our contract model is based on previous work in agricultural supply chains, a very complex kind of VOs [21, 16]).

There are several proposals for contract specification. Some of them are designed for specific purposes where the domain of negotiatable items is predefined, like SLAs (e.g.,[37]). This is not our approach. More generic contract specification approaches need an expressive language to describe the commitments agreed among the partners. Several of them use logic-based approaches, like [72, 47, 43, 69, 88]. Our approach is different. It is was designed to be used in real agricultural supply chains where the participants are autonomous and heterogeneous. Thus, the process of designing a contract template must be feasible by a team of one TI professional and a lawyer.

A contract expresses commitments among partners. We advocate that a contract in the context of an agricultural supply chain and VOs should express the agreement among more than two partners. However, most of the ones proposed in the literature are bi-lateral, just a few are multi-party, e.g., [88].

Contracts are the outcome of some negotiation process which may be done with some level of software assistance. We proposed automatic negotiation performed by software agents and guided by contract templates [16]. Other proposals also use templates, e.g., [49, 22, 33]. An alternative for negotiation are matchmaking approaches like [68].

Kallel and others [55] propose a multi-agent negotiation model for a particular contract type of a specific supply chain. The negotiation model consists of a heuristic negotiation protocol and a decision-making model. The authors' approach diverge from ours in two aspects. Firstly, they understand a supply chain as a neighborhood: a focused company, its suppliers and its customers. We consider a supply chain "from farm to fork". To the limit, an alliance (i.e., VO) within a specific supply chain would comprise partners of all supply chain levels. A business process of this wide VO would aim at a complex and long-term event, e.g., the organization of the 2014 World Cup (Brazilian people are not used to eat potatoes in every meal; thus, it would be necessary to increase potato crops timely). Secondly, we propose a generic negotiation protocol rather than a specific one. This prevents us to aim at optimizations (e.g., maximizing profits), but widens the protocol applicability.

Pitt and colleagues [75] propose a voting protocol for multi-agent VOs. This voting protocol characterises the powers, permissions, obligations and sanctions of the voters and is specified by means of Event Calculus. This protocol is used in the context of an agent community. Decisions must be taken during the life-cycle of such a community. The authors' approach is slightly different from ours. They do not focus on establishing an agreement (i.e., contract) for future enactment, but on decisions taken "on-the-fly" during the enactment.

A number of authors use contracts to describe the coordination of activities of partners, e.g., [86, 60]. Others use contracts a means of monitoring the fulfillment of the contract's commitments, e.g., [88]. In addition, [6] use a Petri net-base approach to discuss how a partner should implement its part of the contract complying to the contract's description.

Most of research efforts that combine VOs and contracts are in the context of agent societies, e.g., [85, 91]. They use contracts to shape the agents' behaviour, i.e., the actions they might or might not be undertaken regarding to the use of shared resources. They are not business contracts indeed.

5.7 Conclusion

Contracts can be used to assemble and manage VOs. Multi-party contracts are required in such a context, because a set of bi-lateral contracts can destroy or hide relationships among the partners. Such relationships are hard to model and manage, because they are not homogeneous within a VO. Some issues may demand consensus among the partners. Others, competition among them or exactly two partners must reach a consensus. Thus, it is important that different styles of marketplaces be seamlessly combined in a single coherent assembling process. This paper presented how SPICA negotiation protocol can be used to do so. It also outlined some details of the protocol's implementation.

Future work includes the implementation of an infrastructure for a mechanism to monitor the contract's fulfillment.

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

Capítulo 6

Implementation

E. Bacarin, E.R.M. Madeira, C.B. Medeiros and W.M.P. van der Aalst. SPICA's Multi-party Negotiation Protocol: Implementation using YAWL. *Intl. J. of Coop. Info. Sys.*, submitted, 2009.

Abstract
A supply chain comprises several different kind of actors that interact either in an ad hoc fashion (e.g., an eventual deal) or in a previously well planned way. In the latter case, how the interactions develop is described in contracts that are agreed on before the interactions start. This agreement may involve several partners, thus a multi-party contract is better suited than a set of bi-lateral contracts. If one is willing to negotiate automatically such kind of contracts, an appropriate negotiation protocol should be at hand. However, the ones for bi-lateral contracts are not suitable for multi-party contracts, e.g., the way of achieving consensus when only two negotiators are haggling over some issue is quite different if there are several negotiators involved. In the first case, a simple bargain would suffice, but in the latter a ballot process is needed. This paper presents a negotiation protocol for electronic multi-party contracts which seamlessly combines several negotiation styles. It also elaborates on the main negotiation patterns the protocol allows for: bargain (for peer-to-peer negotiation), auction (when there is competition among the negotiators) and ballot (when the negotiation aims at consensus). Finally, it describes an implementation of this protocol based on Web services, and built on the YAWL Workflow Management System.

6.1 Introduction

Face-to-face negotiations are being increasingly replaced by electronic negotiations (e-negotiation) in all situations in which interactions among partners are dynamic and must follow the "just-in-time" principle. Such a negotiation process gives rise to electronic contracts (e-contract) that are enacted by a suitable supporting system and produce data that can be used to assess in which extension their partners are fulfilling the contract's provisions, allowing for early corrective actions. Electronic negotiations, either assisting human negotiators or automating the whole process, facilitate the interaction among parties and speed up contract construction.

eBay is a prime example of such a situation, but it presents just a facet of the multiple challenges to be faced. One can extend this kind of scenario to a situation where multiple kinds of enterprises need to interact and negotiate distinct issues, at different levels. One typical example of such a complex situation is contract negotiation within supply chains.

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 [65]. It is composed of distributed, heterogeneous and autonomous elements, whose relationships are dynamic.

Efficiency and profitability within a supply chain depends on several factors, including the speed and flexibility with which the participants arrange their relationship in terms of goals and commitments, and their capability of assessing how these goals and commitments have been reached or fulfilled.

A contract negotiation differs from a single item (or single bundle) negotiation, which is more common. In the latter, partners haggle over the item's price and configuration and, if they agree, two actions follow: the buyer hands over money to the seller, and the seller hands over the item to the buyer.

Conversely, a contract states rights and duties among partners (and also subsets thereof). It mainly comprises a set of actions or intended effects that, if accomplished, would result in the fulfillment of these duties or rights. In this perspective, a negotiation comprises taking into consideration this set of actions and haggling over specific attributes that qualify them (e.g., configuration, price and quality constraints). A contract is a future plan and contains a number of statements of intention.

Multi-party contracts are a special kind of contract. The negotiation process must be able to express to whom each duty or right is applicable to.

e-Negotiation and contract management are subject to intensive research. However, most papers concentrate on one single kind of negotiation protocol (e.g., auction, bargain). Moreover, implementation issues are handled either at the protocol (communications) level or concern negotiation logics. In [16] we discuss SPICA Negotiation Protocol

(SPICA, for short), a new kind of contract e-negotiation in which each clause may be established according to a different negotiation protocol. This provides the flexibility needed in multi-party negotiation in which partners change at each situation (e.g., typical of supply chains). In [17] we present how SPICA's protocols can be combined to build virtual organizations. This paper details our implementation of SPICA which shows innovative ways of solving the challenges in implementing combined negotiation styles for e-contracts:

1. we present how multi-protocol issues can be attacked by means of specific negotiation patterns, which can then be instantiated at will, dynamically.

2. we discuss how to take advantage of a workflow engine – YAWL – to implement the middleware that orchestrates interactions among partners. The negotiation protocol is seen as a workflow in which each interaction (e.g., an offer) is considered a task the partner negotiator must carry out (e.g., decide whether to accept the offer or not). In addition, YAWL's workflow engine is extended with a tailor-made service which takes care of the information exchanged among negotiators.

The main contributions of this paper are: (a) it presents a negotiation protocol for electronic contracts which combines three basic negotiation styles: bargain (for peer-to-peer negotiation), auction (when there is competition among the negotiators) and ballot (when the negotiation aims at consensus); (b) it depicts the main negotiation patterns the protocol allows for; (c) it describes an implementation built on a WfMS (YAWL).

The paper is organized as follows. Section 6.2 presents an overview of YAWL. Section 6.3 describes briefly SPICA's contracts and the SPICA negotiation protocol. Section 6.4 presents the main negotiation patterns supported by the protocol. Section 6.5 discusses the protocol's implementation, including a negotiation execution. Section 6.6 reviews related work. Finally, section 6.7 concludes the paper.

6.2 YAWL Overview

In this paper, we use YAWL (Yet Another Workflow Language) to realize and support the SPICA protocol. YAWL was developed after a rigorous analysis of existing workflow management systems and related standards using a comprehensive set of workflow patterns [7]. YAWL is both a language and a system supporting this language [52]. There are three main reasons for using YAWL. First of all, the language is simple yet much more expressive than most other languages. Since a wide range of workflow patterns are directly supported, it is easy to quickly realize complex workflows. Second, the YAWL system has rigorously adopted a service-oriented architecture. This makes it easy to use

YAWL in a distributed setting where multiple parties and also software from multiple vendors need to cooperate. Finally, YAWL is well-grounded, i.e., right from the start formal semantics and analysis techniques were provided. (Unlike most other languages where semantics and analysis are more of an afterthought, rather than a first priority.)

This section introduces the YAWL language, the supporting system, and some of the more advanced capabilities used in the implementation of SPICA. For more details, the reader is referred to [52].

6.2.1 YAWL: A Language Based on Patterns

In the area of workflow one is confronted with a plethora of products (commercial, free and open source) supporting languages that differ significantly in terms of concepts, constructs, and their semantics. One of the contributing factors to this problem is the lack of a commonly agreed upon formal foundation for workflow languages. The workflow patterns initiative [7] aims at establishing a more structured approach to the issue of the specification of control flow dependencies in workflow languages. Based on an analysis of existing workflow management systems and applications, this initiative identified a collection of patterns corresponding to typical control flow dependencies encountered in workflow specifications, and documented ways of capturing these dependencies in existing workflow languages. These patterns have been used as a benchmark for comparing process definition languages and in tendering processes for evaluating workflow offerings. See `http://www.workflowpatterns.com` for extensive documentation, flash animations of each pattern, and evaluations of standards and systems.

While workflow patterns provide a pragmatic approach to control flow specification in workflows, Petri nets provide a more theoretical approach. Petri nets form a model for concurrency with a formal foundation, an associated graphical representation, and a collection of analysis techniques. These features, together with their direct support for the notion of state (required in some of the workflow patterns), make them attractive as a foundation for control flow specification in workflows. However, even though Petri nets support a number of the identified patterns, they do not provide direct support for the cancellation patterns (in particular the cancellation of a whole case or a region), the synchronizing merge pattern (where all active threads need to be merged, and branches which cannot become active need to be ignored), and patterns dealing with multiple active instances of the same activity in the same case. This realization motivated the development of YAWL [5] (Yet Another Workflow Language) which combines the insights gained from the workflow patterns with the benefits of Petri nets. It should be noted though that YAWL is not simply a set of macros defined on top of Petri nets as the expressiveness is increased considerably (see [7, 5, 52] for discussions on this).

Before describing the architecture and implementation of the YAWL system, we introduce the distinguishing features of YAWL relevant to this paper. As indicated in the introduction, YAWL is based on Petri nets. However, to overcome the limitations of Petri nets, YAWL has been extended with features to facilitate patterns involving multiple instances, advanced synchronization patterns, and cancellation patterns. Moreover, YAWL allows for hierarchical decomposition and handles arbitrarily complex data.

Figure 6.1 shows the modeling elements of YAWL. At the syntactic level, YAWL extends the class of workflow nets described in [1] with multiple instances, composite tasks, OR-joins, removal of tokens, and directly connected transitions. YAWL, although being inspired by Petri nets, is a completely new language with its own semantics and specifically designed for workflow specification.

A *workflow specification* in YAWL is a set of *process definitions* which form a hierarchy. *Tasks*[1] are either *atomic tasks* or *composite tasks*. Each composite task refers to a process definition at a lower level in the hierarchy (also referred to as its decomposition). Atomic tasks form the leaves of this graph. There is one process definition without a composite task referring to it. This process definition is named the *top level workflow* and forms the root of the hierarchy of process definitions.

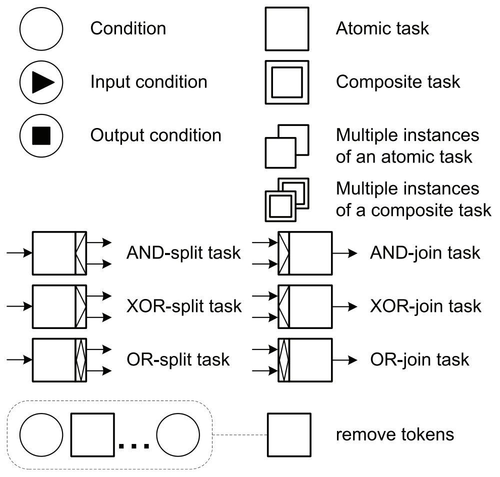


Figura 6.1: Symbols used in YAWL [5].

Each process definition consists of *tasks* (whether composite or atomic) and *conditions* which can be interpreted as places. Each process definition has one unique *input condition* and one unique *output condition* (see Figure 6.1). In contrast to Petri nets, it is possible to connect 'transition-like objects' like composite and atomic tasks directly to each other

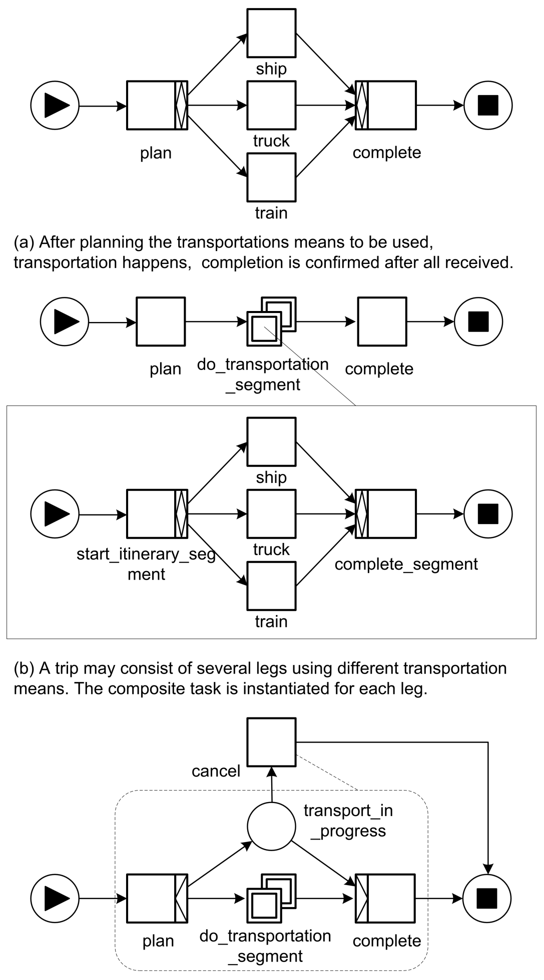
[1]We use the term *task* rather than *activity* to remain consistent with earlier work on workflow nets [1].

without using a ‘place-like object’ (i.e., conditions) in-between. For the semantics this construct can be interpreted as a hidden condition, i.e., an implicit condition is added for every direct connection.

Both composite tasks and atomic tasks can have multiple instances as indicated in Figure 6.1. We adopt the notation described in [1] for AND/XOR-splits/joins as also shown in Figure 6.1. Moreover, we introduce OR-splits and OR-joins corresponding respectively to Pattern 6 (Multi choice) and Pattern 7 (Synchronizing merge) defined in [7]. Finally, Figure 6.1 shows that YAWL provides a notation for removing tokens from a specified region denoted by dashed rounded rectangles and lines. The enabling of the task that will perform the cancellation may or may not depend on the tokens within the region to be “canceled”. In any case, the moment this task completes, all tokens in this region are removed. This notation allows for various cancellation patterns.

To illustrate YAWL we use the three examples shown in Figure 6.2. The first example (a) illustrates that YAWL allows for the modeling of advanced synchronization patterns. Consider that, within an agricultural supply chain, a given load of coffee, packed in bags, is to be transported between two warehouses in two different cities. This amount can be divided in up to three partial loads and transported by different means of transportation. Task *plan* is an ‘OR-split’ (Pattern 6: Multi-choice) and task *complete* is an ‘OR-join’ (Pattern 7: Synchronizing merge). This implies that every planning step is followed by a set of transportation tasks *ship*, *truck*, and/or *train*. It is possible that all three transportation tasks are executed, but it is also possible that only one or two tasks are performed. The YAWL OR-join synchronizes only if necessary, i.e., it will synchronize only the transportation tasks that were actually selected, signaling that the full amount of coffee bags were received at the destination.

Figure 6.2(b) illustrates another YAWL specification of the same stage of the supply chain. In contrast to the first example, a transportation trip may include multiple legs, i.e., an itinerary between the two warehouses may include multiple segments. Typically, transportation between two consecutive legs is performed by one transportation means. However, the load that arrives at one point may be divided and transported by different transportation means to the next point in the itinerary, where again the load can be rearranged for the next leg. For example, the trip between the warehouses may entail three itinerary segments with distinct characteristics. In the first segment, coffee bags are transported from a farm warehouse to a cooperative by truck. The cooperative collects bags from several farms, and the load is then transported to the docks by train. Finally, the coffee is transported to a final warehouse overseas by ship. Figure 6.2(b) shows that multiple segments are modeled by multiple instances of the composite task *do_transportation_segment*. This composite task is linked to the process definition also shown in Figure 6.2(b). In the case of multiple instances, it is possible to specify upper



(a) After planning the transportations means to be used, transportation happens, completion is confirmed after all received.

(b) A trip may consist of several legs using different transportation means. The composite task is instantiated for each leg.

(c) Again the composite task is instantiated for each leg but now it is possible to cancel the whole trip by removing tokens from the region indicated.

Figura 6.2: Three YAWL specifications (adapted from [2]).

and lower bounds for the number of instances. It is also possible to specify a threshold for completion that is lower than the actual number of instances, i.e., the construct completes before all of its instances complete. The example shows that YAWL supports the patterns dealing with multiple instances (Patterns 12-15). Only few systems support multiple instances.

Finally we consider the YAWL specification illustrated in Figure 6.2(c). Again composite task *do_transportation_segment* is decomposed into the process definition shown in Figure 6.2(b). Now it is however possible to withdraw planned segments by executing task *cancel*. Task *cancel* is enabled if there is a token in *transport_in_progress*. If the environment decides to execute cancel, everything inside the region indicated by the dashed rectangle will be removed. In this way, YAWL provides direct support for the cancellation patterns (Patterns 19 and 20). Note that support for these patterns is typically missing or very limited in existing systems.

In this section we illustrated some of the features of the YAWL language while focusing mainly on the control-flow. We did not discuss the data aspects – each workflow execution involves several variables (over 10 variables in the example of coffee transportation) – because they are not needed to understand the rest of the paper, even though variable specification and instantiation is an essential aspect in workflow execution. It is important to note that YAWL also supports many resource patterns [78], data patterns [79], exception patterns, flexibility patterns, service interaction patterns, etc.

6.2.2 YAWL System

The YAWL language is supported by a full-fledged workflow management system. Figure 6.3 shows the architecture of YAWL. Like any workflow management system, YAWL has an *Engine* and *Process Designer*. The *Process Designer* is used to construct YAWL specification using the notations described before. These can be verified and once they contain no error, they are runable and the engine can instantiate instances of such models. The engine also records events in so called event logs and persists data beloning to running instances. The *Resource Service* is used to offer or push work to human resources. This is just one example of a service that can be invoked from YAWL. There are many other *YAWL Services* and Figure 6.3 only shows a fraction of the available services. The main design principle of the YAWL System is that the *Engine* should be completely agnostic with regards to the services interacting with it. The *Engine* is unaware of the inside behavior of services, i.e., services can be seen as a black box that subcontract work. Moreover, YAWL itself can be seen as a service, e.g., one YAWL engine can subcontract work to another engine. This makes YAWL truly service-oriented system and highly suitable for supporting the SPICA protocol. For more details, we refer to [52].

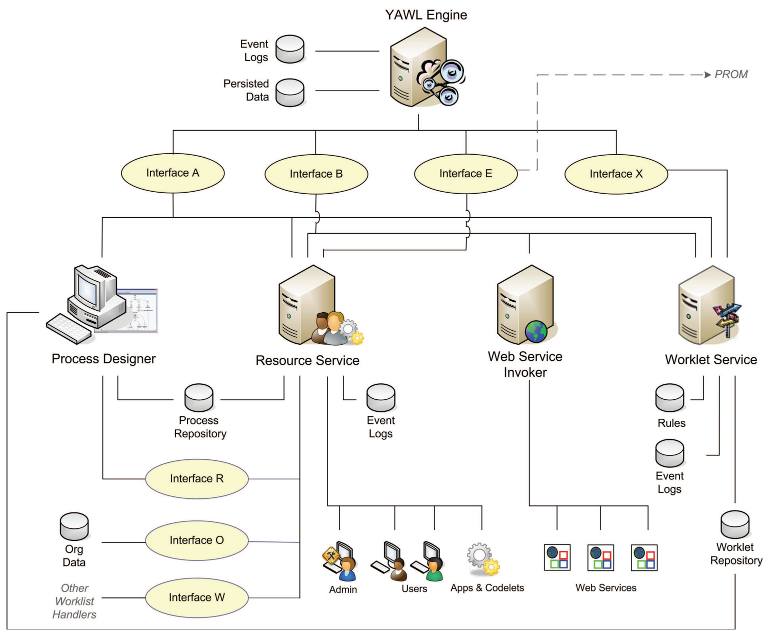


Figura 6.3: The architecture of YAWL [52].

6.3 SPICA's contracts and negotiation protocol: an overview

This section highlights some basics about SPICA's contracts and negotiation protocol. For simplicity's sake some details were omitted. The complete description is presented in [16].

Usually, a negotiation process consists of determining the price for an item. Our negotiation process is more general, i.e.: (a) the items to negotiate may involve values other than prices or numbers. Thus, we prefer to think of the "best value", instead of the "highest" ("lowest") value. Sometimes, we use the typical negotiation jargon in the broad sense, e.g., the phrase "pay more" would mean to offer a better value (in a negotiator's perspective); (b) the negotiation process may concern something other than a physical item (e.g., a requirement or a quality criterion to be met). Thus, we use the term "goods" in this broader sense.

This section is organized as follows. Section 6.3.1 reviews briefly the format of SPICA's contracts and the roles run by the negotiation partners. Section 6.3.2 overviews the main data exchanged by the negotiators within a negotiation process. Section 6.3.3 presents the negotiation messages that convey such data and shape the interactions among the negotiators. Section 6.3.4 describes the overall approach we adopted to implement SPICA'S negotiation framework on top of YAWL system.

6.3.1 The Contract and the Actors

Figure 6.4 depicts a class diagram for our contracts. A contract is an instance of a contract model. A contract model is composed of clauses. A clause may have one or more properties. A property is an attribute to be negotiated. It has a name (which is unique in the contract) and may appear in more than one clause. The negotiation process aims at assigning values to properties.

A property is negotiated once. If it appears in more than one clause, once negotiated, its value holds for all occurrences. There are two kinds of properties: simple properties and compound properties. A *simple property* holds a scalar value. A *compound property* is a vector of scalar values and each entry corresponds to a different partner.

The partners in a contract are identified by unique names. Each clause may have two sets of partner names: the *authorized* partners and the *obliged* partners. An obliged partner must perform some action to produce the intended result. Authorized partners have the right to receive such a result.

Consider, for instance, the clause below. Two parties – a farm and a storage provider – engage a negotiation process over this clause. They will haggle over the storage space in cubic meters (property `QC`) and the price (property `PC`) of each cubic meter. The properties `OBLIGED` and `AUTHORIZED` are assigned with the names of the parties who agreed on the values for `QC` and `PC`. Note that, in this example, there is only one obliged negotiator (the storage provider) and only one authorized negotiator (the farm). Note also that the names of these (simple) properties are preceded by $ and properties that refer to partner names are preceded by @.

```
The party @OBLIGED agrees to make available $QC cubic meters at a
cost of $PC per cubic meter.
```

The next example is quite similar to the previous one. Now, several storage firms (St 1, St 2, etc) promise to offer space to the farms. Each firm will make available a different amount of space at different prices. Note that all properties in this example are compound and, thus, are preceded either by `@@` or `$$`.

```
The party @@OBLIGED agrees to make available $$QC cubic meters at
a cost of $$PC per cubic meter.
```

Conversely, in case space has the same price for all firms, but each firm provides a different amount of space, the clause would have been written like (note that there is only one `$` before `PC`):

```
The party @@OBLIGED agrees to make available $$QC cubic meters at
a cost of $PC per cubic meter.
```

The actors in a negotiation setup are the *negotiators* and the *notary*. There is a special kind of negotiator – the *leader* – who orchestrates the negotiation. Every negotiation setup has at least two negotiators (the leader is always present). More negotiators are possible. Unlike other contract models, SPICA supports several kinds of negotiation styles for the negotiation of a single contract. This is made possible by defining typical *negotiation patterns* — e.g., auction and ballot (see section 6.5). The notary is a trustworthy third-party. It mediates some negotiation patterns (e.g., ballots) and is responsible for building the contract instance after a successful negotiation. Some negotiators may be *proxy negotiators*, i.e., they represent a group of negotiators. The negotiators in such a group do not take part directly in the negotiation process, but they will always be represented by the proxy negotiator. However, the proxy negotiator cannot sign the contract (only the negotiators it represented can sign).

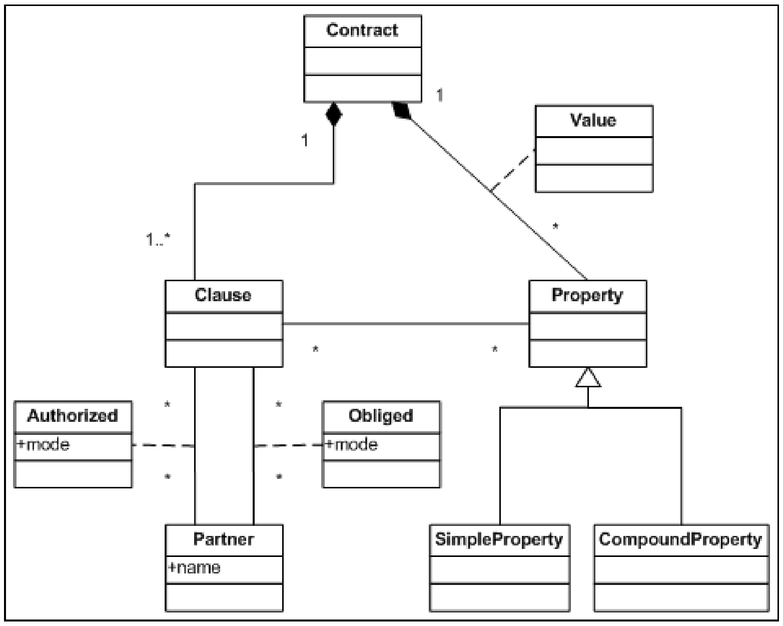


Figura 6.4: *Contract.*

6.3.2 Main Data Types

The negotiation process begins with a setup phase. In this phase, all necessary setup operations are performed, such as: the contract model to be negotiated is determined and a new negotiation instance is created, the negotiators register in. Its main outcome is twofold. First, it creates a new negotiation instance that is distinguished by an identifier. This identifier is named `nid` throughout the paper and used in virtually all messages exchanged within that negotiation process. Second, the negotiators and their roles are determined. Negotiators have distinct names and credentials. The name identifies the negotiator and is used to address messages. The *credential* states its capabilities in the negotiation process, e.g., some negotiators may have veto power in a ballot.

The negotiation of a contract's properties may involve several negotiation rounds. Each round runs one of the possible negotiation styles (bargain, ballot, auction), aims at assigning values to the properties of a particular clause, and establishes who are the obliged and the authorized partners regarding such a clause. The actual negotiation style, the obliged and benefited partners and other attributes that configure the negotiation round are described by means of *negotiation descriptions*, which are represented by `nd` throughout this paper.

The properties to be negotiated – i.e., the subject of the negotiation – are not included in a negotiation description, rather, they are described by means of an RFP (request for proposal), an offer, an RFI (request for information) or an Info (information).

An RFP is an invitation. A negotiator A sends an RFP to a negotiator B asking for a value for one or more properties. RFPs are represented by either `rfp` or `r( )`throughout the paper.

An offer is a promise. Typically, it is an answer for an RFP. A negotiator answers an RFP by sending back an offer that confirms the values of predefined properties and proposes values for the desired properties. The offer must comply with the restrictions indicated in the RFP. The negotiator that issued an offer is committed to it. Offers are represented by `offer` or `o( )` throughout the paper.

An RFI is similar to an RFP: it requests values for properties. In addition, it also requests lower and upper bounds for them. An RFI is answered by an Info. An Info is similar to an offer; however, the negotiator who issues an Info is not committed to it. An RFI is represented by `rfi` and an Info by `info`.

RFPs, offers, RFIs and Infos have also other data elements, such as, unique identifiers respectively, `rid` (for RFPs), `oid` (for offers), `riid` (for RFIs), and `iid` (for Infos); the credential of the negotiator who has created it, and the list of names of the receiver negotiators. The credential for a particular negotiator, say `n1`, is represented by `!n1` in the figures presented in the paper.

6.3.3 Message types

Tables 6.1 and 6.2 present the types of messages that are provided by SPICA Negotiation Protocol and the attributes they convey. They are used in the negotiation patterns described in Section 6.4. There are also a few extra messages needed by the framework's implementation. They are called *control messages* and will be presented later. Most of these messages convey a few common parameters, namely: the sender name (`from`) that can be used to address a response message; the identification of the negotiation instance (`nid`) the message corresponds to, and the negotiation description (`nd`).

Request Proposal. It communicates an RFP within the context of a particular negotiation style (described by `nd`).	`Rp(from,nid,nd,rfp)`
Request Information. It communicates an RFI.	`Ri(from,nid,nd,rfi)`
Request Agreement. It communicates an offer.	`Ra(from,nid,nd,offer)`
Request Auction Step. The leader asks the notary to conduct an auction step. An English auction step is described by an RFP and a Dutch auction step is described by means of an offer.	`Ras(from,nid,nd,rfp)` `Ras(from,nid,nd,offer)`
Request Ballot. The leader asks the notary to conduct a ballot process. The issue to be voted is described by (`ballot`)	`Rb(from,nid,nd,ballot).`
Request Vote Agreement. The notary requests a vote from a negotiator. The expected answer is "agree" or "disagree". (`bid`) is the ballot instance.	`Rva(from,nid,nd,bid,ballot)`
Request Vote Preference. The notary requests a vote from a negotiator. The expected answer is a value for a property.	`Rvp(from,nid,nd,bid,ballot)`

Tabela 6.1: Negotiation messages (requests).

As an introductory example consider Figure 6.5, suppose that a farm (`F`) would bargain over the freight fee (`ff`) with a transportation company (`TC`) to deliver coffee bags to the docks. In this scenario, `F` will send an RFP to TC by means of a `Rp` message asking `TC` how much it would charge the transportation (i.e., it asks a value for `ff`). This message conveys four attributes. The first holds the name the sender used to register in the negotiation. The receiver uses this name to address a reply message. The second (`nid`) is the negotiation instance identification. The third attribute describes the details of the current negotiation style (a bargain, in this example). Each negotiation style has specific attributes within this description, but all of them share a common subset. These three attributes are present in most of the negotiation messages. The last attribute is an RFP that asks a value for `ff`.

`TC` receives this message, analyses the received RFP and answers it by sending back an offer that assigns a value to `ff`. This offer is conveyed by means of an `Ra` message.

Answer for Request Proposal. An `Rp` message can be answered by two exclusive messages: `Ra` sends an offer in response to the previous RFP; (`Ino`) declines the invitation.	`Ra(nid,nd,offer)`
	`Ino(from,nid,rid)`

Answer for Request Information. An `Ri` message can be answered by two exclusive messages: `Ari` sends an Info in response to the previous RFI; (`Ini`) informs that the negotiator will not provide the asked information.	`Ari(from,nid,info)`
	`Ini(from,nid,riid)`

Answer for Request Agreement. An `Ra` message can be answered by three exclusive messages: `Aa` agrees on the proposed offer; `Ad` disagrees on it; another `Ra` proposes a counter-offer.	`Aa(from,nid,nd,offer)`
	`Ad(nid,nd,offer)`
	`Ra(from,nid,nd,offer)`

Answer for Request Auction Step. Two consecutive messages answers a Request Auction Step (`Ras`) message: The notary sends an `Aas` message to the leader to inform the leader that the requested auction step will be performed (conversely, `Nas` to refuse doing so); at the end of the auction step, the notary sends an `Ica` message to the leader with all received bids.	`Aas(nid,aucid,rid)`
	`Aas(nid,aucid,oid)`
	`Ica(nid,aucid, offer_lst)`
	`Nas(from,nid,rid)`
	`Nas(from,nid,oid)`

Answer for Request Ballot. Notary returns two consecutive messages in response to a `Rb` message. First, the message `Ab` acknowledges the leader that it will conduct the requested ballot (conversely, `Nb` refuses it). At the the end, the notary returns to the leader the ballot's result by means of an `Ibr` message.	`Ab(from,nid,bid,rid)`
	`Ab(from,nid,bid,oid)`
	`Ibr(from,nid,bid,bresult)`
	`Nb(from,nid,rid)`
	`Nb(from,nid,oid)`

Answer for Request Vote Preference. An `Av` answers a previous `Rvp` message. The voter sends the notary its vote, abstention, or veto. This message conveys the voter's credential (`crd`), used for checking, e.g., if the voter has veto power.	`Av(from,nid,bid,crd,vote)`
Answer for Request Vote Agreement. The same `Av` message, but the only allowed alternatives are: abstention, `ok` (i.e., agree), `nok` (i.e., not agree), and veto (if applicable).	`Av(from,nid,bid,crd,vote)`

Tabela 6.2: Negotiation messages (answers).

6.3.4 Implementation Overview

The participants are Web services that use the messages presented in the previous section (Section 6.3.3) to develop the intended negotiation (see [16]). Thus, sending a negotiation message to a participant means invoking a particular operation of a service. Thus, in principle, each participant (web service) can make sure it complies with the protocol rules and the negotiation can proceed without any external help. However, there are a few chores that would be better undertaken by a tailor-made middleware, providing a clear separation between the participant's negotiation strategy implementation and message transportation.

We use the YAWL WfMS (Workflow Managemente System) as such a middleware. It is used to model and to execute the protocol. In this model, a workflow task typically represents a message received or to be sent. Thus, when a message is received by the middleware, the corresponding task is enabled. This causes the execution engine to invoke a specific routine (i.e., a Web service operation) to handle such a message. Handling a

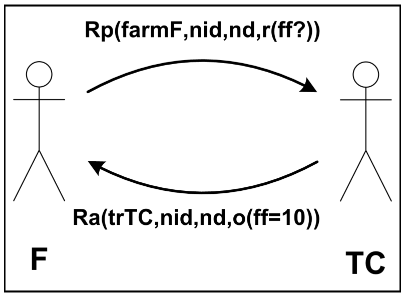


Figura 6.5: *A negotiation message and an answer.*

message means analysing it and sending back an answer for it. Thus, there must be another task already enabled responsible for receiving such a response. For this purpose, these tasks are, in general, arranged in pairs. See Figure 6.6. The first task (`T1d`) receives a negotiation message (e.g., `Rp`). The underlying workflow system instrumented with additional software validates the message, dispatches it to the intended negotiator (`Negotiator 1`) and, simultaneously, enable the second task by sending it a an internal *control message* (ACK, in this case). The second task (`T1f`) is responsible for receiving the negotiator's answer (e.g., `Ra`) and forwarding it to the next task (e.g., `T2d`). Thus, tasks like the first one are *dispatch-like tasks* and like the second one are *forward-like tasks*. Tasks `T2d` and `T2f` have similar arrangement. This kind of arrangement occurs frequently in the patterns presented in Section 6.4.

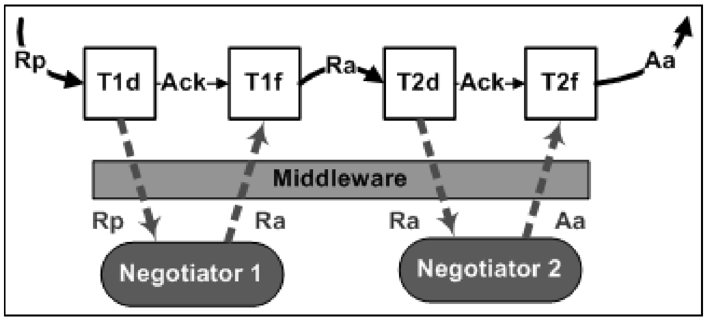


Figura 6.6: *Tasks arranged in pairs.*

6.4 Description of Patterns

This section presents several negotiation patterns supported by the SPICA protocol.

Each pattern is composed of several sections (Figure 6.7). *Pattern name* and *Known as* are used to identify the pattern. The *Motivation* part elaborates on the goal and presents the context of the pattern. This part uses common concepts and does not use SPICA protocol's parlance. The *Problem description* part presents the problem to be handled by the pattern in terms of SPICA's concepts. *Problem Solution* details a possible solution to the problem, i.e., it shows how the message types (described in Section 6.3.3) can be combined to achieve the expected results and how the participants interact. *Implementation of Solution* shows the pattern realization in YAWL and addresses some specific setups or additional elements needed for the presented solution.

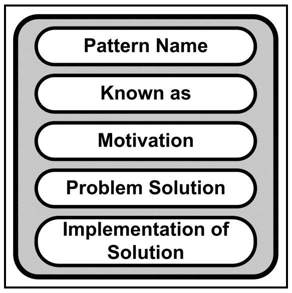


Figura 6.7: *Sections of a pattern.*

6.4.1 Pattern name: *Bargain* Known as: -

Motivation: The simplest style of negotiation is a bargain. Two negotiators haggle over a value to be settled (the subject of the negotiation). They usually exchange counter-offers, e.g., between a set of farms and a transportation company

Problem Solution: Bargains are characterized by the following pattern. A leader starts a bargain requesting a proposal from the other negotiator. The leader and the negotiator exchange offers and counter-offers until the negotiation comes to a conclusion (either by an acceptance or final disagreement message). Figure 6.8 shows such a pattern, where the offer and counter-offers appear in messages 2,3 and 4. In more detail, the figure shows five messages exchanged between the leader (`ld`) and a negotiator (`n1`). It starts **(1)** when the leader (`ld`) sends a request proposal message (`Rp`). The negotiator responds **(2)** with either (a) an offer conveyed by a request agreement message (`Ra`), or (b) a non-offer message (`Ino`) declining to take part of this bargain. If the negotiator has decided to take part, it sends an `Ra` message with an initial proposal. The negotiation carries on. **(3)** The leader does not agree with the proposal and sends a counter-offer (another `Ra` message). The negotiator (4) sends another counter-offer. This situation is repeated until one of the participants (in this case, the leader) comes to a conclusion finishes the process by either (a) agreeing by means of an agreement message (`Aa`), or (b) disagreement message (`Ad`).

Let us now examine the parameters in each message. **(1)** When the leader requests a proposal from Negotiator `n1` (message `Rp`), the parameter (nid) identifies the negotiation instance, created during the negotiation setup (not shown in pattern). The second parameter (tuple `d`) is negotiation description. It informs that this negotiation concerns a bargain (`BG`), the clause being negotiated (`cj`),the obliged party list (`ol`), and the authorized party list (`al`). The third parameter describes the proposal requested by means of an RFP (tuple `r`). Note that the first and second parameters happen in all pattern's messages.

The RFP's parameters are the following. The value `r1` identifies the RFP. An RFP has two set of properties: one pre-assigned by the originator (`pas`) and one set of asked properties (`aps`). Finally, `rt` describes the restrictions (i.e., a boolean expression) on the expected answer. An RFP also has the identification of the originator and of the receiver(s); they are implied by the arrows in all patterns presented in the paper. Here, the originator is the leader `ld` and the receiver is negotiator `n1`.

In the counter-offer cycle started at **(2)**, the partners use `Ra` messages to exchange offers (tuple `o`, in the third parameter). The offer, identified by `oi1`, correlates to the starting `Rp` message (`r1`). The offer must also mention and assign all the properties of the corresponding RFP. Thus, it contains the same assignment in the previous RFP (`pas`) and assigns values for all the asked properties (`aps`). This set of assignments is represented by `assgn1` in the figure. The offer also repeats the RFP's restrictions (`rt`). The last but one parameter means that the negotiator `n1` agrees with this offer (because it has created the offer itself), and the last parameter means that the offer has not been evaluated by the receiver. Like an RFP, an offer also identifies the originator and the receivers, implied by the arrow as well. **(3)** The leader does not agree with the proposal and sends a counter-offer (another Request Agreement message). It is similar to the previous one with different assignment for the asked properties. Note that the counter-offer correlates to the previous offer (`oi1`), instead of the starting RFP (`r1`). **(5)** (a) When a negotiator (the leader in this example) agrees on an offer, the agreement message (`Aa`) contains exactly the same offer (`assgn3`) the leader has received (which it agrees upon); however the last but one offer's parameter contains the identification of both the leader and the negotiator (since both agree upon `assgn3`) and the last parameter (`A`) states that the offer was agreed upon. Conversely, (b) a disagreement message (`Ad`) to the other negotiator is similar to the `Aa` message, but the last parameter that is set to `D`.

Implementation of Solution: The pattern implementation in YAWL is shown in Figure 6.9. In this figure, arrows correspond to messages and boxes to tasks. Tasks labeled with `L` inside a pentagon represents tasks performed by the leader and the ones

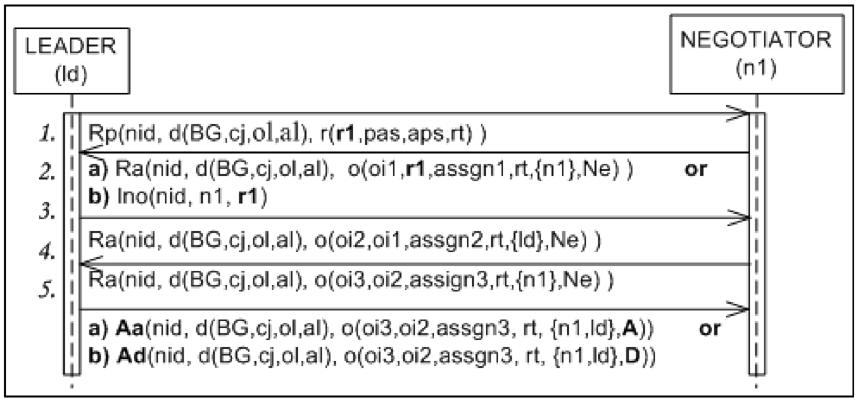

LEADER (ld)
NEGOTIATOR (n1)
1. Rp(nid, d(BG,cj,ol,al), r(r1,pas,aps,rt))
2. a) Ra(nid, d(BG,cj,ol,al), o(oi1,r1,assgn1,rt,{n1},Ne)) or
b) Ino(nid, n1, r1)
3. Ra(nid, d(BG,cj,ol,al), o(oi2,oi1,assgn2,rt,{ld},Ne))
4. Ra(nid, d(BG,cj,ol,al), o(oi3,oi2,assign3,rt,{n1},Ne))
5. a) Aa(nid, d(BG,cj,ol,al), o(oi3,oi2,assgn3, rt, {n1,ld},A)) or
b) Ad(nid, d(BG,cj,ol,al), o(oi3,oi2,assgn3, rt, {n1,ld},D))

Figura 6.8: *Bargain interaction pattern.*

labeled with a `Ng` inside a circle are performed by a negotiator (other than the leader). This process starts at the topmost circle (with a triangle inside) and finishes at the bottommost circle (with a square inside).

An offer and an RFP have a valid lifespan. A late offer is only discarded. Note that the tasks on the left are related to the leader and the ones on the right relate to the negotiator.

An `Rp` arrives at the task `Negotiator Received RFP`. The task is accomplished by *dispatching* the `Rp` message to the intended negotiator. The task's result is an `ACK` message sent to the task `Negotiator Creates Offer` (enabling it). The negotiator uploads an answer via the infrastructure. The infrastructure executes this task by matching the `ACK` message and the answer provided by the negotiator and *forwarding* the answer to one of `RFP Declined` or `Received Negotiator Offer` or `Received Notification from Negotiator`, depending on the negotiator's reply.

This two-task arrangement appears on most of the interactions in this net. Dispatch-like tasks are identified by a little triangle on the bottom-leftmost corner of its label (e.g., `Negotiator Received RFP`) and the forward-like ones have a small mark on its top-rightmost corner (`Negotiator Creates Offer`). However, some sent messages do not demand a response (e.g., `Aa`. In this case, there is only the dispatch-like task (e.g., `RFP Declined`) and its resulting ACK message is just discarded.

6.4.2 Pattern name: *English Auction* Known as: *Ascending Auction*

Motivation: The subject of the negotiation is in dispute by several negotiators, e.g., the several coffee farms. Supposedly the negotiator who is more willing to get the item will offer a better value for it.

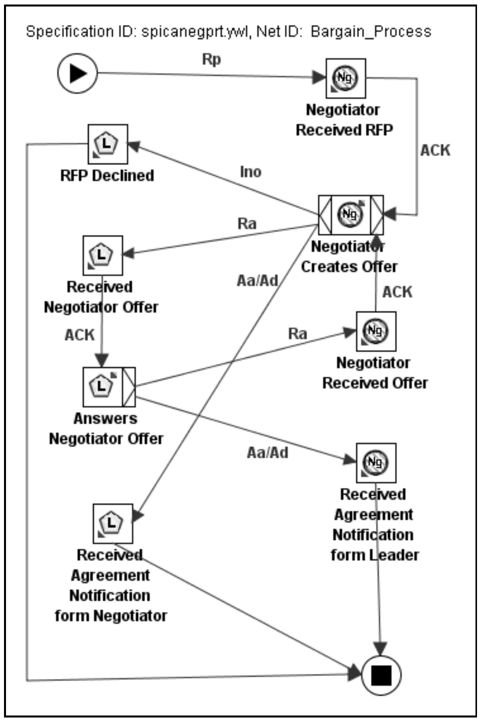

Specification ID: spicanegprt.ywl, Net ID: Bargain_Process
Rp
Negotiator Received RFP
ACK
RFP Declined
Ino
Ra
Negotiator Creates Offer
Received Negotiator Offer
Aa/Ad
ACK
ACK
Ra
Negotiator Received Offer
Answers Negotiator Offer
Aa/Ad
Received Agreement Notification form Leader
Received Agreement Notification form Negotiator

Figura 6.9: *Bargain process (YAWL)*

Problem Solution: In Figure 6.10, there are a leader (`ld`) and a notary (`nt`). There are several bidders that are represented by `n+`. One individual bidder is represented by `nk`. An auction is characterized by the following pattern.

The auction is controlled by the leader and helped by the notary. It develops in auction steps. In one auction step (a) the leader requests the notary to broadcast the subject of the auction to the bidders and to accept the corresponding bids. The auction's subject defines restrictions over the bids (e.g., a minimum price). (b) the bidders send bids to the notary. (c) The notary collects a few bids and sends them all to the leader, finishing the auction step. The leader may either agree on one or more bids received in the last or any previous steps or request the notary to run another step. In the latter case, the leader may increase the restrictions over the expected bids (e.g., increase the minimum price).

Let us examine the dynamics of the auction pattern presented in Figure 6.10. **(1)** The leader asks the notary to announce the auction (message `Ras`). The first two parameters, as usual, are the negotiation identification and the negotiation description. The negotiation description for an auction is similar to the one of a bargain, but conveys a few extra attributes: the list of the negotiators that take part in the auction (`{n1,n2,...}`), the number of expected bids (`max_answr`)), and the time interval the notary should wait for the expected bids (`tmout`). These last two parameters indicate that each auction step has a predefined lifespan and may receive many bids (typically, it is only one). Aged or excess bids are only discarded. The third parameter conveys the RFP that describes the property(ies) to be auctioned. Typically, such an RFP imposes a restriction on the asked properties (e.g., a minimum acceptable price). **(2)** The notary signals the leader that it will control the auction (message `Aas`) and informs the identifier for this auction step (`a1`) and the RFP's identification (`r1`). **(3)** The notary broadcasts the negotiation description and the RFP to all negotiators. The broadcast is represented by a stair-shaped arrow. **(4)** Each negotiator answers this request by (a) sending an offer to the notary (message `Ra`), or (b) informing the notary that it is not interested in the current auction step (message `Ino`). The offer has a suitable assignment for each asked property (not shown). The last two parameters are shown. They state that the negotiator `nk` agrees with this offer and that the offer was not evaluated by the counter-party (`Ne`). **(5)** The notary collects these offers and sends them back to the leader by means of an `Ica` message. The last parameter is the list of received offers. It does not include the `Ino` messages sent in step 4b (if any). **(6)** The leader chooses the best offer(s) (the *step winner(s)*) according to its own criterion, and starts another auction step with a more restrictive RFP (e.g., a higher minimum acceptable price). This cycle repeats and **(7)** eventually no negotiator proposes an offer (the leader receives an empty list of offers). **(8)** The leader agrees with the best offer(s) of the previous round (identified by `oy`) and **(9)** may disagree with all defeated offers (or simply let them age).

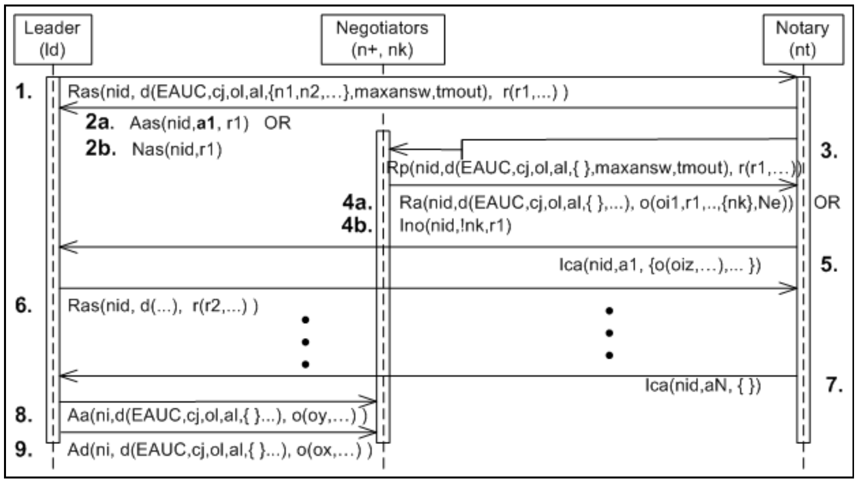

Leader (ld)
Negotiators (n+, nk)
Notary (nt)
1.
Ras(nid, d(EAUC,cj,ol,al,{n1,n2,...},maxansw,tmout), r(r1,...))
2a. Aas(nid,a1, r1) OR
2b. Nas(nid,r1)
3.
Rp(nid,d(EAUC,cj,ol,al,{ },maxansw,tmout), r(r1,...))
4a. Ra(nid,d(EAUC,cj,ol,al,{ },...), o(oi1,r1,..,{nk},Ne))
OR
4b. Ino(nid,!nk,r1)
Ica(nid,a1, {o(oiz,...),... })
5.
6.
Ras(nid, d(...), r(r2,...))
Ica(nid,aN, { })
7.
8.
Aa(ni,d(EAUC,cj,ol,al,{ }...), o(oy,...))
9.
Ad(ni, d(EAUC,cj,ol,al,{ }...), o(ox,...))

Figura 6.10: *English auction interaction pattern.*

Implementation of Solution: Consider Figure 6.11. It models both English and Dutch auctions. Dutch auction is described in Section 6.4.4. The leader's tasks are on the left, the notary's are in the middle, and the negotiator's (bidder's) are on the right. A new auction request (`Ras`) arrives at task `Requested Auction Step`. It is dispatched to the notary. The notary may either refuse or accept conducting this auction step. In the first case, it answers an `Nas` message to leader. In the second case, the notary starts an internal timer to control the lifespan of the initiating auction step and answers two simultaneous messages: `Aas` to the leader accepting the job and `Rp` to the bidders announcing the new auction step. To model such a behaviour and due to the fact that `Aas` and `Rp` will not be uploaded by the notary exactly at the same point in time, but in any order, `Requested Auction Step` has a AND-split that sends an `ACK` to `Notary Decides I` and `Notary Decides II` enabling them to receive all those three messages. Let us discuss what happens in either cases.

Recall that upon receiving the `ACK` message, both tasks `Notary Decides I` and `Notary Decides II` are enabled. In the first case (the refusal), the `Nas` message may reach either of them and, as a result, control is diverted to `Notary has Refused`, which dispatches the message to the leader.

Conversely, the notary might have accepted. In this case, control heads for `Start New Auction Step`. It is just a synchronization task (AND-join) that waits for the arrival of both messages `Aas` and `Rp`. It directs the message `Aas` to task `Notary has Acknowledge` and message `Rp` to task `New English Auction`. It relies on non-trivial XQueries to direct the received messages to the right tasks. It also enables task `Collected Bids` by sending the same `ACK` as of `Requested Auction Step`. Recall that Section 6.3.4 observed that an arc itself does not convey any data. This enabling is a clear example: task `Collected Bids` gets enabled by the arc originated from `Start New Auction Step`, but obtains its input data of a local variable assigned at the completion of task `Requested Auction Step`.

Task `New English Auction` dispatches the `Rp` message to the bidders and sends an `ACK` to task `Waiting Bids`. Whenever `Waiting Bids` receives a bid, it forwards the bid to task `Receive Bids`. This task dispatches the bid to the notary and re-enables `Waiting Bids` allowing it to receive other bids from other negotiators. The notary stores the received bids in its private database.

When the auction step's lifespan has elapsed, the notary collects the bids it has received, creates an `Ica` message containing the received bids and uploads it to the engine by means of the `Collected Bids` task. This message is forwarded to task `Leader Waiting Bids`, which dispatches it to the leader. Finally, the leader can start a new auction step (`Ras`) or agree with a few bids (message `Aa`) and disagree with the defeated ones (message `Ad`). These messages are dispatched to the corresponding negotiators by task `Receive (Dis)Agreement`. The bottommost arc is used when there was no bid at all (auction failed) or it was a Dutch auction. Note that tasks `Notary has Acknowledged` and `Receive Bids` do not have their forward-like counter-parts. This happens because the received message do not demand an answer. For more details the reader is referred to [20].

This model uses two cancellation regions: for tasks `Notary has Refused` and `Collect Bids`. The first region comprises tasks `Notary Decides I` and `Notary Decides II` and their outgoing arc. Recall that both tasks were simultaneously enabled, but `Nas` message reaches one of them. This cancellation region makes sure the other task is canceled. The second region comprises all tasks and arcs on the right of task `Collected Bids`, disabling the reception of late bids.

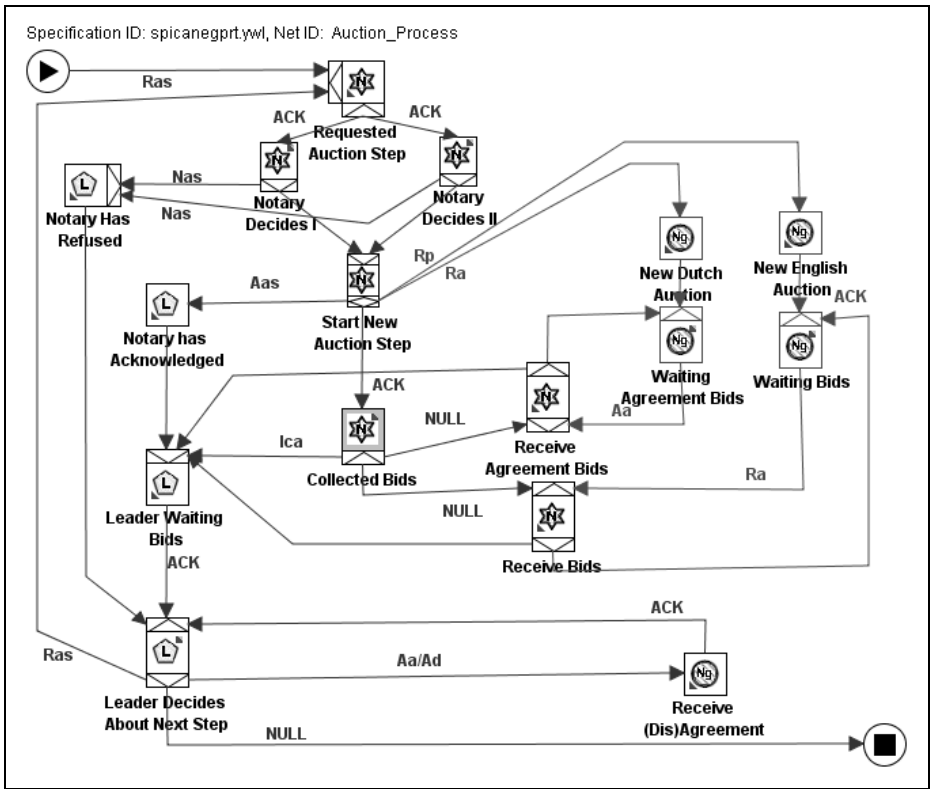


Figura 6.11: *Auction process (YAWL)*

6.4.3 Pattern name: *Open Ballot* Known as: -

Motivation: Some negotiation issues comprise a set of options to be chosen by the negotiators. The goal of the negotiation is to determine the most preferred option.

Problem Description: The ballot's issue may include several properties. Most of the properties' values are assigned in advance. There is exactly one property not assigned and a predefined list of possible values for this property. The ballot's goal is to assign the most preferred value to this property.

Problem Solution:

This pattern, shown in Figure 6.12. There are a leader (`ld`) helped by the notary (`nt`) and several negotiators, all of them represented by `n+`. An individual negotiator is represented by `nk`. **(1)** The leader asks the notary to conduct the ballot process (`Rb` message). This message conveys three parameters: the negotiation instance identification (`nid`), the negotiation description (tuple `d`) and the ballot description (tuple `b`). The negotiation description informs that this negotiation concerns a ballot (`BLT`). Besides the common attributes, it also conveys the list of voters (`{n1,...}`). The leader is not always a voter: if so, it will be included in this list. The negotiation description includes other parameters that specify the dynamics of the ballot, all represented by `xp`, such as: how much time to wait for votes, the number of votes needed to approve the issue, the minimum number of votes to an alternative be elected, how to handle ties (e.g., by considering the leader's vote twice, considering a tie as an approval, etc).

The ballot description comprises an RFP (tuple `r`, showing only the RFP identifier) and the set of alternatives (`{a1,...,az}`). This RFP has exactly one unbound property. Thus, voting means choosing one among the alternatives. **(2)** (a) The notary acknowledges the leader that it will conduct the ballot (message `Ab`) and informs the ballot identifier (`bid`). (b) Conversely, the notary refuses this job (message `Nb`) and there is no further interactions. **(3)** If the notary agreed, it broadcasts the ballot's subject to all negotiators (`Rvp`). If the leader is a voter, it also receives this request for voting. The dashed line indicates that it is not always the case. **(4)** A voter sends its vote to the notary (message `Av`): (a) Each vote may choose one alternative; or (b) may be an *abstention*, or (c) a veto (if applicable). A non-authorized veto is considered as an abstention. Note that parameters `nid` and `bid` correlate the vote to the the negotiation instance and to the ballot. The vote also conveys the voter's credential (`!nk`). **(5)** The notary counts the votes (*xi* is the number of votes the alternative *ai* has received) and broadcasts the result. The result may be approved, not approved, or vetoed. The result of a ballot is disclosed by means of `br` data type, written as:

```
br(blst,{ch1:nv,...},ofr)
```

The first parameter is the ballot status, i.e., whether the ballot was approved (`A`), not approved (`Na`) or vetoed (`V`). The second parameter lists how many votes (`nv`) each choice (`chI`) has received. The approved choice is in the beginning of this list. In case of an approved ballot, the last parameter (ofr) conveys an offer that assigns the chosen value to the respective property. Such an offer is agreed by the leader and the notary (the notary is a kind of a proxy for the voters).

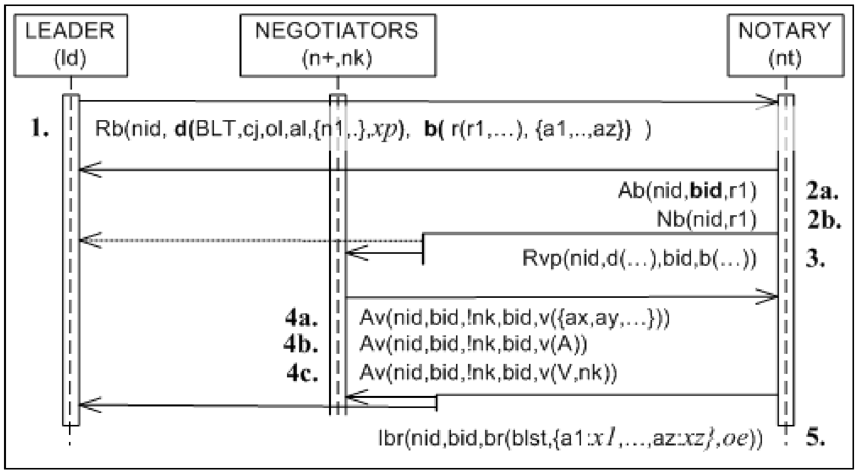

LEADER (ld)
NEGOTIATORS (n+,nk)
NOTARY (nt)
1.
Rb(nid, d(BLT,cj,ol,al,{n1,..},xp), b(r(r1,...), {a1,..,az}))
Ab(nid,bid,r1)
2a.
Nb(nid,r1)
2b.
Rvp(nid,d(...),bid,b(...))
3.
4a.
4b.
4c.
Av(nid,bid,!nk,bid,v({ax,ay,...}))
Av(nid,bid,!nk,bid,v(A))
Av(nid,bid,!nk,bid,v(V,nk))
Ibr(nid,bid,br(blst,{a1:x1,...,az:xz},oe))
5.

Figura 6.12: *Open Ballot interaction pattern.*

Implementation of Solution: Figure 6.13 shows the ballot's implementation. It implements both the open ballot pattern and the close ballot pattern (Section 6.4.4). This net follows the same rationale of the auction's net (Section 6.4.2). However, it is a bit simpler. Note that the auction's net also implements two patterns (English and Dutch auction). The bids for each pattern are semantically and structurally different. Thus, each kind of bids is received in different parts of the auction's net. Conversely, there is no substantial difference between votes of each pattern and they can be handled equally. Another difference is that, whereas the auction step's result is only sent to the leader the ballot's result is broadcasted to all negotiators, including the leader.

In the case of an open ballot, the notary, via task `Start Ballot`, forwards the message `Rvp` to the negotiators via task `Request Vote`. Later, the notary receives the votes enclosed within message `Av`.

6.4.4 Other Patterns

This section briefly presents a few other patterns the negotiation protocol supports.

Dutch auction. In a Dutch auction, one or more properties are in dispute by several negotiators (bidders). The auctioneer announces successive (typically, decreasing) assignments for such property(ies). There is a predefined time span between successive announcements. The negotiator that first agrees upon the current value wins the auction.

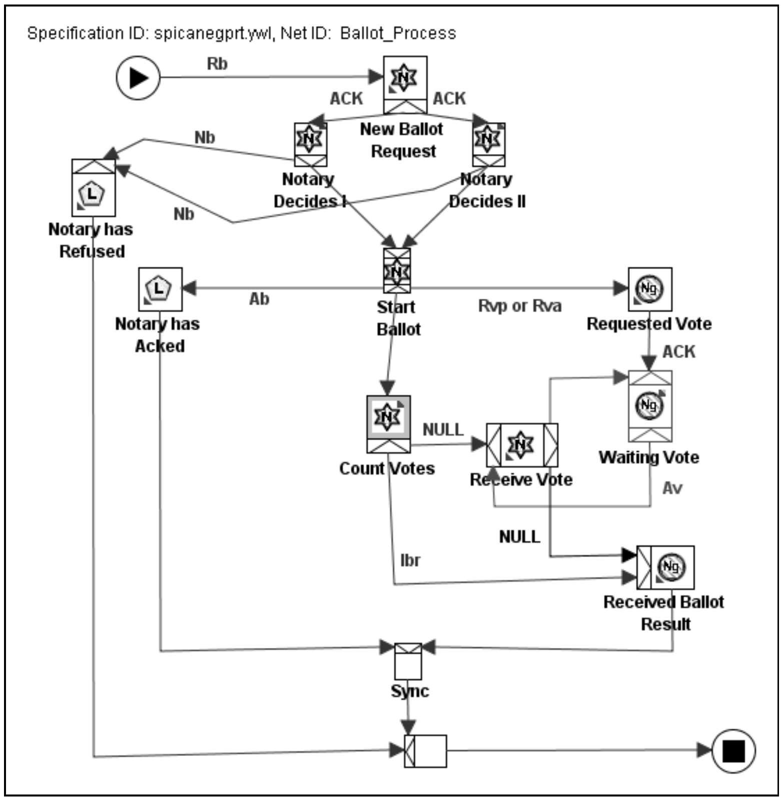


Figura 6.13: *Ballot process.*

This pattern is quite similar to the English auction pattern (Section 6.4.2). However, each auction step is described by an offer conveyed by message `Ra` (see Figure 6.11) and the bidder agrees with such an offer by means of message `Aa`.

Closed ballot. A closed ballot is similar to the open one (Section 6.4.3). However, the ballot issue is described via an offer and the voters may only agree or disagree on it (abstention and veto are also possible). The ballot's issue is forwarded by the notary via an `Rva` message (Figure 6.13) and the votes are conveyed by an `Av` message.

Sealed bid. A sealed bid is a kind of auction with only one auction step. The auctioneer asks for bids and chooses the best bid. The implementation of this pattern is quite similar to the bargain pattern: the auctioneer broadcasts an RFP (message `Rp`) to all bidders and waits for the bids during a predefined time span. The bidders send offers, via `Ra` messages, in response. Then, the auctioneer agrees on the best offer.

Request for Information. All the presented patterns can be augmented by a number of requests for information (RFIs) sent by the leader before starting the proper negotiation. We have also defined and implemented this pattern, this factoring out a common type of request found in all negotiation stages.

6.5 Middleware Implementation

The previous sections explained SPICA's negotiation messages (Section 6.3) and how they can be combined into different negotiation styles which are modeled as workflows (Section 6.4). Such messages are transported by means of a middleware (recall Figure 6.6) and delivered at specific interfaces of a few Web services (negotiator's and notary's). This middleware comprises YAWL's workflow engine which we instrumented with tailor-made software (hereafter called NS Service). The engine executes the workflow and invokes NS to handle the transported messages. Sections 6.2 presented YAWL system and Section 6.3.4 pointed out a few implementation issues needed to understand design decisions. This section details the protocol's implementation.

SPICA's negotiation protocol is modeled as a YAWL model comprising several subnets. Some of them were presented in Section 6.4. Each subnet has a number of tasks that are responsible for handling the messages exchanged among the negotiators. A task has a *decomposition*. The decomposition defines its *input* and *output variables* as well as a so-called *YAWL service*. This is a Web service and can be seen as a procedure called by the workflow engine to process the input variables and return a set of results (the outputs). Thus, the engine can forward these results to subsequent tasks. The format of the messages are described in Section 6.5.1. SPICA's specific decompositions are described in Section 6.5.2. Section 6.5.3 presents the tailor-made custom service (NS Service).

6.5.1 Message Format

Section 6.3.3 presented the negotiation messages exchanged within a negotiation process. Messages are XML files sent between tasks. This section shows their layout and also describes a few extra messages, called *control messages*, needed by the infrastructure.

Figure 6.14 shows an excerpt of a message. All messages have a common header. The first piece of information in the header is the message's type (`tp`) – e.g., `Ra`, `Rp`. The second piece of information (`posted`) records when the message was posted to the middleware, while the third one (`expires`) determines its lifespan, i.e., how long the sender will wait for an answer. The fourth piece of information (`expectansw`) is a boolean value that instructs SPICA's infrastructure whether it must expect a reply for such a message. Finally, `fromname` and `receivers` record the message's originator and recipients. Following this header there is a long `choice` element with an entry for each possible message type. Only the first two appear in Figure 6.14. Such elements encode the message's parameters (presented in Section 6.3.3)

```
<complexType name=‘‘MessageType’’>
  <sequence>
    <element name=‘‘tp’’ type=‘‘MsgType’’/>
    <element name=‘‘mid’’ type=‘‘MsgIDType’’/>
    <element name=‘‘pid’’ type=‘‘MsgIDType’’/>
    <element name=‘‘posted’’ type=‘‘dateTime’’/>
    <element name=‘‘expires’’ type=‘‘dateTime’’/>
    <element name=‘‘expectansw’’ type=‘‘YesNoType’’/>
    <element name=‘‘fromName’’ type=‘‘string’’/>
    <element name=‘‘receivers’’ type=‘‘ToType’’/>
    <choice>
      <element name=‘‘rp’’ type=‘‘MsgReqProposalType’’/>
      <element name=‘‘ra’’ type=‘‘MsgReqAgreementType’’/>
       Similar for other types of messages...
    </choice>
  </sequence>
</complexType>
```

Figura 6.14: *Negotiation Message Format.*

Variable	**Mode**	
RecM	In & Out	The received message. The received message can be a negotiation message or a control message.
ExpectAnsw	In Only	The types of the expected reply messages, i.e., the outgoing messages in RespM1.
RespM1	Out Only	A task can generate an outgoing message

Tabela 6.3: NSTask decomposition.

Figure 6.15 depicts the encoding of the parameters of an `Rp` message – see Section 6.3.3. Other messages follow similar arrangement.

Additional so-called control messages were introduced and are used internally by the middleware, e.g., ACK, NULL.

6.5.2 Decompositions

Most of the tasks related to negotiation interactions have a common decomposition called `NSTask`. Table 6.3 summarizes the parameters of this decomposition.

The first two variables provide input for the task and the last is an outgoing message. `RecM` is also an output variable, which just outputs the received message. `RecM` may be a negotiation message or a control message. The YAWL service associated to a task of such a decomposition is the NS service.

Recall the dual task arrangement (i.e., the dispatch-like and forward-like tasks ar-

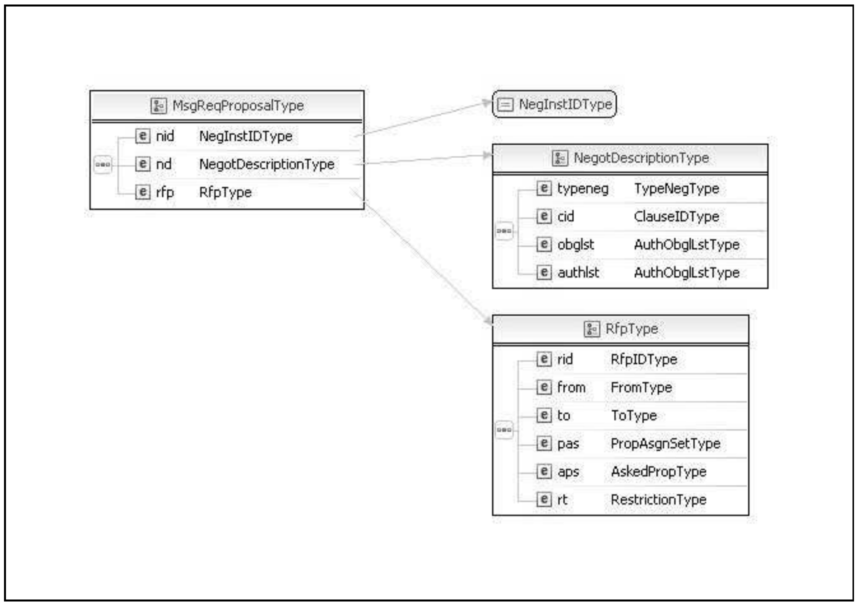


Figura 6.15: *Parameters of an RP message: MsgReqProposalType.*

rangement) presented in Section 6.3.4 and the auction subnet presented in Figure 6.11. Figure 6.16 shows this subnet in detail. A request for a new auction step (`Ras`) message arrives at `Requested Auction Step` via its `RecM` input variable. This task is a dispatch-like task. The `Ras` message is dispatched to the notary and an `ACK` control message is returned via `RespM1` output variable and assigned to a local variable (named `NewAucAck`). This `ACK` message contains information that will relate the received `Ras` message to the corresponding answers to it. The `ACK` message is received at `Notary Decides I` via its `RecM` input variable. This task is a forward-like task. In response to the `Ras` message, the notary has to send three different messages: **(1)** an `Aas` message to the leader; **(2)** either an `Rp` (English auction) or an `Ra` message (Dutch auction) to the negotiators, and **(3)** an `Ica` message to the leader at the end of the auction step. Thus, the notary sends these messages (only `Aas` is shown in the picture) to NS service which: a) verifies if the incoming responses are within the list of `ExpectAnsw` input variable (offending messages are discarded), b) correlates them to the previous `ACK` message and c) checks them into the workflow engine. The output message will be assigned to output variable `RespM1`. For instance, Table 6.4 shows the values for these parameters for the first two tasks in Figure 6.11. Note that the values assigned to `RespM1` must be in the `ExpectAnsw` input variable.

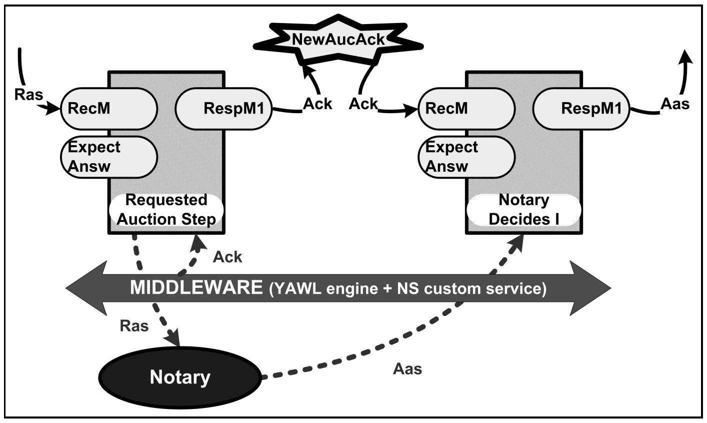


Figura 6.16: *Tasks with decomposition detailed.*

Task	**RecM**	**ExpectAnsw**	**RespM1**
Requested Auction Step	Ras	ACK	ACK
Notary Decides I	ACK	Aas, Rp, Ra	Aas

Tabela 6.4: Examples of assignment of tasks' variables.

6.5.3 NS Custom Service

Tasks are processed by means of associated YAWL services. The YAWL workflow engine implements a few predefined services and provides support for the implementation of user-tailored services, the so-called *YAWL custom services.* A custom service is a Java class that complies with specific interfaces. The negotiation framework implements a custom service called NS (for Negotiation custom Service) that handles the negotiation messages.

Figure 6.17.a shows the rationale behind a generic custom service and Figure 6.17.b shows how such a rationale is applied in our solution. Consider Figure 6.17.a, the standard YAWL solution for custom services. When **(1)** a task is enabled and its input parameters (IP) are available (typically from a local variable), **(2)** the engine invokes the associated custom service (Csrv). This service (2a) performs the so-called "check out" operation via a specific interface provided by the engine. This causes the task to be in the "executing" state. The service obtains the input data, processes it and (2b) returns the result to the engine by means of a "check in" operation. This causes the task to be completed and **(3)** the result to be assigned the output parameters (OP). In the end, other tasks may be enabled and the results may be available for further processing. To sum up, it is a synchronous arrangement.

On the other hand, SPICA's messages, depicted in Figure 6.17.b, are asynchronous.

The negotiators are web services that implement some specific interfaces and are not aware of YAWL's engine. In the figure, **(1)** A negotiation message (IM) is sent to a negotiator (e.g., an Rp message). **(2)** This message enables task T' and is transfered to its input variable `RecM`. The engine invokes NS that (2a) checks out the incoming message, (2b) immediately checks in a specific ACK control message, (2c) enabling the subsequent task (T"). At the same time, (2d) NS dispatches the incoming message to the intended negotiator. **(3)** The negotiator analyzes the received message and (3a) sends an answer to NS via the NS' `checkInIF` interface. Meanwhile, the ACK message has arrived to task T". NS is invoked by the engine and (3b) checks out the ACK message. (c) NS matches the ACK and answer messages (see Section 6.5.2), perform a few validation procedures and checks in the messages which are assigned to the output variable `RespM1`. This outgoing message (OM) is available for processing by the subsequent tasks.

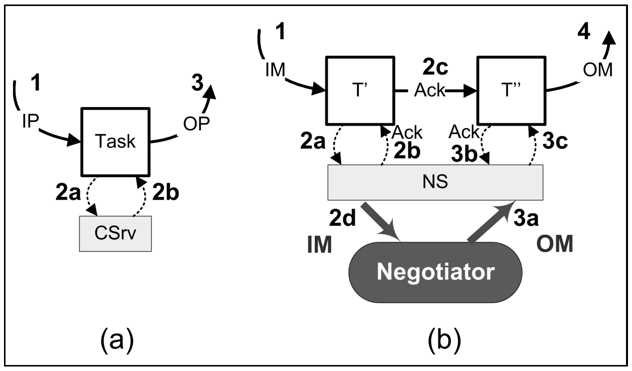


Figura 6.17: *Dispatch of negotiation messages.*

Figure 6.18 goes into details about the interaction between the YAWL engine, the NS service and a negotiator from the latter's perspective.[2] It shows that NS provides two interfaces: one to receive requests from the YAWL engine (interface `B'`) and other to receive requests form a negotiator via the *checkInIF* interface (`Ci`). There is also a setup interface (interface `S`) that is out of the scope of this paper. Every negotiator (NM) implements a set of specific interfaces (described in [16]) depicted as gray circles. Operations of these interfaces are activated when the corresponding negotiation message is dispatched to the negotiator. There is an extra interface (black circle) for exceptions (out of the scope of this paper). The negotiator also uses a *communication adaptor* (COM ADP) to upload an answer message.

A YAWL custom service extends the so-called `InterfaceBWebSideController` abstract class (interface `B'` in Figure 6.18) and implements a method named `handle-`

[2]The rationale is the same for the notary.

`EnabledWorkItemEvent`. When a task is enabled, this method is invoked by the YAWL engine. A custom service interacts (i.e., checks out) with the engine through the so-called interface `B`. Using this interface, the custom service gets the received message, uploads the corresponding ACK message, and uploads the corresponding answer message.

Let us examine Figure 6.18 in detail. When a task whose decomposition is NSTask is enabled, **(1)** the engine calls the method `handleEnabledWorkItemEvent` within the interface `B'`. This method receives a so-called *work item* as a parameter. Such work item keeps quite a few pieces of information about the enabled task. **(2)** NS service interacts with the engine and gets the message received by the task, i.e., the content of input variable `RecM`, validates the message and **(3)** checks in an `ACK` message. **(4)** According to the type of the received message, NS service calls an appropriate operation of one of these interfaces, i.e., it dispatches the negotiation message. **(5)** The negotiator interacts with NS via the `Ci` interface. In response to a negotiation message, the negotiator uploads the answer messages to NS. Recall that the negotiator is not aware about NS service or about YAWL engine. The message uploading is intermediated by means of a communication adaptor (COM ADP). This adaptor mimics a negotiator by implementing the same interfaces. When a negotiator calls one of its operations, it creates a negotiation message in the format described in Section 6.5.1 and forwards it to NS. If another middleware was used, only the adaptor should be replaced. Finally, **(6)** NS correlates the uploaded answers with the corresponding ACK message and checks in the corresponding task. To do so, NS uses a few data contained within the received work item. The uploaded answer is assigned to this task's output variable (RespM1). Details about the validation and correlation processes are presented next.

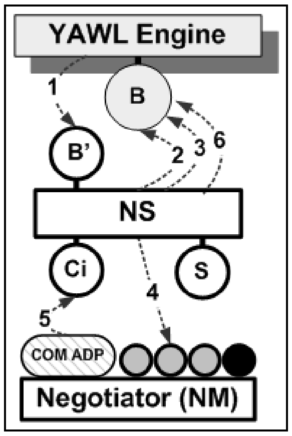


Figura 6.18: *NS service from the perspective of one negotiator.*

The algorithm used to dispatch messages uses two data structures: a table of dispatched messages (`TDM`) and a queue of not processed incoming messages (`NpMsgs`). There is an instance of them to each different negotiation instance. The table `TDM` has 8 columns:

- `mid`: it is the identification of the dispatched message `M`.
- `pid`: M's parent message identification
- `nansw`: it keeps the number of answers the message has received so far (-1 means the message has not been dispatched; 0, the message was dispatched, but no answers have arrived so far; other positive numbers stand for the number of received answers),
- `ndscd`: it counts the discarded answers for a specific message
- `expires`: it indicates the message's expiration date
- `posted`: Date when the message was posted into NS service.
- `msg`: it keeps the message itself for auditing purposes.
- `ackLst`: recall that when message M is dispatched, `ACK`s are sent to subsequent tasks enabling them to receive answers for M. Thus,`ackLst` is a list that contains the work items of the tasks enabled by such `ACK`s.

In what follows, `TDM[nid,mid].expires` means the expiration date of message `mid` in the negotiation `nid`; `M.mid()` means the message identifier for message `M`.

An incoming message, viz., a message received via the NS' `checkInIF` interface, can only be processed if its enabling `ACK` has already been checked out. If this is not true, the incoming message is inserted in the `NpMsgs` queue and processed upon `ACK` arrival.

Figures 6.19, 6.20 and 6.21 depict algorithms used by NS to handle incoming messages. Figure 6.19 shows the actions executed by NS when a negotiation message (i.e., not an `ACK` message) is received via the `RecM` input variable (Figure 6.18.b, step 1). Note the created `ACK`: it keeps the identification of its respective message (`ACK.pid`, step b3) that will be used to correlate with the expected answer (to arrive).

Conversely, if an `ACK` message is received via `RecM`, the algorithm presented in Figure 6.20 is executed. The received `ACK` is used to match a dispatched message (algorithm Figure 6.19) with a future answer. However, it may happen that the answer itself arrives before its `ACK`. In this case, the early answer will have been put in the queue of not processed messages (algorithm Figure 6.21). Thus, step (d) checks if this is the case.

Finally, Figure 6.21 shows the actions executed when a reply message is received via NS' `checkInIF` interface. Such reply message might have arrived earlier that its respective `ACK`. It is checked in step (b). If so, the incoming message is added to the list of

```
a. Check out negotiation message M (Figure 6.17.b, step
2a)
b. Create an new ACK message:
    i.ACK.nid:= M.nid();
    ii.ACK.mid:= new_identifier();
    iii.ACK.pid:= M.mid();
c. TDM[M.nid(),M.mid()].nasw:= -1:  ,
TDM[M.nid(),M.mid()].ndscd:=-1;
  TDM[M.nid(),M.mid()].pid:= M.pid() ;
TDM[M.nid(),M.mid()].msg:= M;
  TDM[M.nid(),M.mid()].expires:= M.expires();
d. Check in the ACK message (Figure 6.17.b, step 2b)
e. Dispatch message M to the appropriate negotiator's
interface. (Figure 6.17.b, step 2d)
```

Figura 6.19: *Negotiation message received via RecM input variable*

```
a. Check out the ACK message (A). (Figure 6.17.b, step 3b)
b. TDM[A.nid(),A.pid()].answ = 0;
TDM[A.nid(),A.pid()].ndscd=0
c. Insert A to TDM[A.nid(),A.mid()].ackLst d. Check
NpMsgs. If there is already a response (R) related to this
ACK message (i.e., R.pid()=A.pid()), take the response off
the queue and execute steps (c) and (d) of the algorithm in
Figure 6.21
```

Figura 6.20: *ACK message received via RecM input variable*

not processed messages. Recall that a message may have several answers. For instance, the message `Ras` at the beginning of an auction (Figure6.11) may receive two answers when the auction step starts (namely, `Aas` and `Rp`) and another message at its end (an `Ica`). In this case, the `Ras`'s `ackLst` will contain three work items respective to tasks `Notary Decides I`, `Notary Decides II` and `Collected Bids` that are responsible for receiving those answers. Note that all such answers have the same parent. Thus, it must be determined which is a suitable work item for the upcoming message: this suitable work item is the one whose list of expected answers contains the type of the upcoming answer. Note that step (c) validates the upcoming message before it is check in within the context of the suitable work item.

```
a. NS receives the response R for previous negotiation
message P via checkInIF interface.  It examines the
response and finds the identification of the parent message
(Figure 6.17.b, step 3a)
    pnid:= R.nid(); pmid:= R.pid())
b. If the respective ACK has not been checkout yet (i.e.,
TDM[R.nid(),R.pid()].nansw =-1):
    i. Insert the response R into NpMsgs
    ii. Exit
c. If R is aged (TDM[R.nid(),R.pid()].expires <
date_posted(R)), or is not valid or violates some
restrictions:
    i. Send exception to the sender.
    ii. Increment ndscd counter and discard the response.
d. If R is not aged:
    i. TDM[R.nid(),R.pid()].nansw++
    ii. Checks in message R. (Figure 6.17.b, step 3c)
```

Figura 6.21: *Response message (R) received via checkInIF interface.*

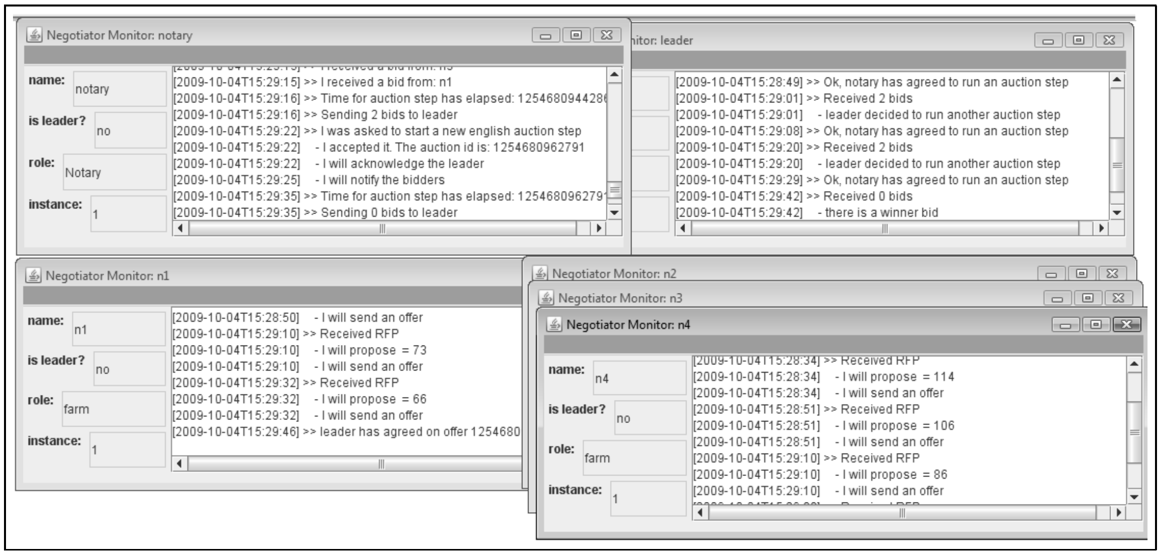
6.5.4 Example Execution

Figure 6.22 shows part of the output of a negotiation instance. Each participant has a window where it prints out a few remarks about the steps and decisions it has undertaken.

A complete execution is shown in another paper ([17]). It presents how the main negotiation styles are combined seamlessly to build a virtual organization.

6.6 Related Work

Modeling. According to [13], a supply chain is a set of disparate members who are



```
name: notary
is leader? no
role: Notary
instance: 1

[2009-10-04T15:29:15] >> I received a bid from: n1
[2009-10-04T15:29:16] >> Time for auction step has elapsed: 125468094428
[2009-10-04T15:29:16] >> Sending 2 bids to leader
[2009-10-04T15:29:22] >> I was asked to start a new english auction step
[2009-10-04T15:29:22]    - I accepted it. The auction id is: 1254680962791
[2009-10-04T15:29:22]    - I will acknowledge the leader
[2009-10-04T15:29:25]    - I will notify the bidders
[2009-10-04T15:29:35] >> Time for auction step has elapsed: 125468096279
[2009-10-04T15:29:35] >> Sending 0 bids to leader
```

```
[2009-10-04T15:28:49] >> Ok, notary has agreed to run an auction step
[2009-10-04T15:29:01] >> Received 2 bids
[2009-10-04T15:29:01]    - leader decided to run another auction step
[2009-10-04T15:29:08] >> Ok, notary has agreed to run an auction step
[2009-10-04T15:29:20] >> Received 2 bids
[2009-10-04T15:29:20]    - leader decided to run another auction step
[2009-10-04T15:29:29] >> Ok, notary has agreed to run an auction step
[2009-10-04T15:29:42] >> Received 0 bids
[2009-10-04T15:29:42]    - there is a winner bid
```

```
name: n1
is leader? no
role: farm
instance: 1

[2009-10-04T15:28:50]    - I will send an offer
[2009-10-04T15:29:10] >> Received RFP
[2009-10-04T15:29:10]    - I will propose  = 73
[2009-10-04T15:29:10]    - I will send an offer
[2009-10-04T15:29:32] >> Received RFP
[2009-10-04T15:29:32]    - I will propose  = 66
[2009-10-04T15:29:32]    - I will send an offer
[2009-10-04T15:29:46] >> leader has agreed on offer 1254680
```

```
name: n4
is leader? no
role: farm
instance: 1

[2009-10-04T15:28:34] >> Received RFP
[2009-10-04T15:28:34]    - I will propose  = 114
[2009-10-04T15:28:34]    - I will send an offer
[2009-10-04T15:28:51] >> Received RFP
[2009-10-04T15:28:51]    - I will propose  = 106
[2009-10-04T15:28:51]    - I will send an offer
[2009-10-04T15:29:10] >> Received RFP
[2009-10-04T15:29:10]    - I will propose  = 86
[2009-10-04T15:29:10]    - I will send an offer
```

Figura 6.22: *An example execution.*

dependent on each other to manage resources (such as inventory and information). The common way of describing such a management is the definition of a business process (BP), i.e., a set of activities that, if properly performed, causes the intended effect. Multiple executions of a BP happen within individual, isolated contexts and, in general, each step causes some kind of transformation, be it physical (e.g., raw material becomes manufactured goods) or logical (e.g., information is produced or updated). Workflow Management Systems (WfMS) are a natural choice for modeling and executing BPs.

Modeling a workflow may be a piece of art or a piece of engineering. In the first case, it is a result of the abilities of the modeler. In the second case, the modeler employs different techniques to achieve a proper result. One of those techniques are design patterns. Design patterns have been around since the mid 90s. Aalst and colleagues have applied them to workflows ([3, 7]) and Mulyar ([66]) presented patterns for process-aware information systems.

This paper proposes using workflows as a means of modeling and executing a specific kind of BP: contract negotiations, and presents core negotiation patterns.

Contracts and Negotiation. There are several proposals for contract specification. Some of them are designed for specific purposes where the domain of negotiable items is predefined, like SLAs (e.g.,[37]). Our approach, instead, does not enforce any item domain. More generic contract specification approaches need an expressive language to describe the commitments agreed among the partners. Several of them use logic-based approaches, like [72, 10, 47, 43, 69, 88]. This is not our approach, though we support generic contract specification. The use of specific languages would demand highly trained computer scientists to design the contract to be negotiated. However, the negotiation environment we propose aims at real supply chains where the participants are autonomous, highly heterogeneous and geographically widespread. It would not be realistic to expect that all participants would have the needed skills.

A contract expresses commitments among partners. However, most of the ones proposed in the literature are bi-lateral, just a few are multi-party, e.g., [88]. Again, our generic contract specification and negotiation patterns does not impose any such constraint.

Contracts are the outcome of some negotiation process which may be done with some level of software assistance. We proposed automatic negotiation performed by software agents and guided by contract templates [16]. This is not unique – other proposals also use templates, e.g., [49, 22, 33]. An alternative for negotiation are matchmaking approaches like [68].

Kallel and others [55] propose a multi-agent negotiation model for a particular contract type of a specific supply chain. The negotiation model consists of a heuristic negotiation protocol and a decision-making model. The authors' approach diverge from ours. They understand a supply chain as a neighborhood: a focused company, its suppliers and its customers. We consider the whole supply chain from the original suppliers to the end consumers.

FIPA[3] proposes standards for the interoperation of heterogeneous agents. A few of them resembles our negotiation styles, e.g., *FIPA Contract Net Interaction Protocol*[4] bears resemblance to our bargains. However, they mostly aim at finding an agent to perform a task. The agent will acknowledge that it has accomplished such a task. In contrast, our negotiation protocol aims at producing a contract that will be enacted in the future. Our protocol itself does not include the execution phase. FIPA also proposes standards for Dutch and English Auctions, but "the auctioneer seeks to find the market price of a good".[5] Our approach is more general than that: we seek to determine good values for a set of properties, not restricted to only price.

A number of authors use contracts to describe the coordination of activities of partners, e.g., [86, 60]. Others use contracts a means of monitoring the fulfillment of the contract's commitments, e.g., [88]. In addition, [6] use a Petri net-base approach to discuss how a partner should implement its part of the contract complying to the contract's description. Other authors use contracts in the context of agent societies to shape the agents' behaviour (e.g., [85, 91]), i.e., the actions might or might not be undertaken according to the use of

[3]IEEE's Foundation for Intelligent Physical Agents: http://www.fipa.org/

[4]http://www.fipa.org/specs/fipa00029/SC00029H.pdf

[5]English auction: http://www.fipa.org/specs/fipa00031/XC00031F.pdf; Dutch auction: http://www.fipa.org/specs/fipa00032/XC00032F.pdf

shared resources. These are not business contracts.

Middleware. SPICA Negotiation Protocol is defined as a set of interfaces to be implemented by the negotiators and notary. Since these participants may be distributed, there must be a communication infrastructure that allows them to exchange negotiation messages (that activate such interfaces). Our current choice is an implementation based on Web services supervised by a workflow engine (YAWL). Other alternatives could have been employed, such as: an implementation of OASIS ebXML Messaging Services[6] (e.g., Hermes[7]) or JADE[8] (extended with WADE). The reasons for using YAWL were stated in the beginning of the paper.

6.7 Conclusion

This paper presented the core implementation of the SPICA Negotiation Protocol, which allows arbitrary composition of negotiation primitives, specified by means of patterns. The implementation relies on the use of the YAWL workflow management system and engine, for which we developed specific services to allow the required negotiation flexilibity. We have implemented several real-life negotiation examples for agricultural supply chains.

Our protocol is more generic than others. It allows for different combination of the negotiation primitives resulting in a wide variety of composite negotiation styles. The ones detailed in this paper are just the most important and suitable for illustrating the use of the protocol's primitives. For a detailed example on protocol primitives, the reader is referred to [17], which shows how two hierarchical parallel negotiations are interleaved to produce a contract.

Additional features have been designed, but not implemented so far. Future work comprises implementing them. Such features include grouping properties in a hierarchical way and negotiating each group separately. Another option is to allow the negotiators to express their intentions, restrictions and reasons at different phases of the negotiation. This would not influence the protocol's control flow, but is intended to help the negotiators' decision making process.

Acknowledgements
This paper was partially supported by CAPES/Brazil and by CNPq (WebMAPS and AgroFlow projects) and FAPESP.

[6]http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/cs02/ebms_core-3.0-spec-cs-02.pdf
[7]http://www.freebxml.org/msh.htm
[8]http://jade.tilab.com/

Capítulo 7

Concluding Remarks and Future Work

7.1 Conclusion

The thesis attacked the problem of handling negotiations in agricultural supply chains in a top down approach. It proposed a model for agricultural supply chains that integrates seamlessly the main features of supply chains discussed in the literature. The model considers both static and dynamic aspects. It also investigated lower level negotiation issues, covering two aspects contract specification and support to flexible multi-party e-negotiations. Multi-party contracts are important in this context because several actors of a supply chain may build alliances comprising mutual rights and obligations. A set of bilateral contracts is not well-fitted for such a purpose. The thesis also presents an implementation of the negotiation protocol that builds on Web services and a workflow engine (YAWL).

The main contributions are:

- a model and an architecture for agricultural supply chains, presented in Chapter 3;
- a negotiation process and a multi-party contract format for agreements among the participants of a supply chain, described in Chapter 4;
- a way of using the negotiation process and a contract to build a virtual organization, presented in Chapter 5;
- the implementation of a middleware that supports the negotiation process by means of extending a workflow engine, presented in Chaper 6.

7.2 Extensions

There are several possibilities for extensions including supply chains, negotiation and contracts.

7.2.1 Supply Chains

Supply Chain Modeling can be analysed from different perspectives. The way agronomists, economists, computer scientists and researchers of other areas view a supply chain differs greatly. All of them are relevant and complementary. However, to the best of our knowledge there is not a unifying model. Although SPICA´s model was inspired and designed aiming at agricultural supply chains, there is no reason that hinders its application to other sorts of supply chains.

Under the Computer Science perspective, SPICA´s model lacks a well-defined description format. There is only a convention for graphical symbols (ellipses, boxes, and so on). There should be, for instance, an XML format for storing supply chain descriptions, left for future work.

7.2.2 Negotiations

This thesis addressed a protocol for negotiation. It presents an implementation of a negotiation scenario only to test the protocol dynamics. The negotiators use simple heuristics to decide about received offers. The thesis did not considered the negotiation intelligence. This is a broad research area to be exploited in the future, specially by means of cooperation with researchers from AI.

The legal aspects of a negotiation process are not considered at depth in the thesis. Future work should consider how to lay a negotiation setup in a way that an agreement made in this setup could be legally enforceable.

We have already designed extensions to the negotiation protocol. One of them concerns providing a way for negotiators to communicate with each other during a negotiation process. This would allow for the establishment of alliances among a subset of them. This extension will be implemented shortly. Renegotiation is another feature also left for future work.

7.2.3 Contracts

The thesis proposed a format for electronic contracts. It is situated between two approaches. In one end are the "paper contracts made digital". Human beings can read them, but computers can only store them. On the other end are the contracts described by means of highly sophisticated formal languages, which fits automatic processing. However, very few computer scientists could be able to read and write them, and it is not a practical approach. We claim that a contract in the context of agricultural supply chains should be designed by a team with at least one IT professional and one lawyer.

Although we have claimed that formal languages are not practical, they are quite useful. Future work might address a way of deriving formal representations out of a SPICA contract. This would help approaching monitoring and auditing issues, which were not addressed by this thesis.

Contracts are instances of contract models. Contracts should be renegotiable. Moreover, the model and the instance may evolve over time. Thus, version control is an aspect that could be further investigated.

Contract execution was not addressed in this thesis. It is related to the execution of coordination plans. In fact, they will be executed by the same mechanism. Contract execution and plan execution were left for future work.

7.2.4 Other Components for SPICA

Most of SPICA´s components were not addressed by this thesis. There is a huge amount of research and implementation efforts to be done concerning chain execution, traceability, summary and regulation management, to name but a few.

To sum up, there is a great amount of further research and implementation to be undertaken, to produce a seamlessly supply chain framework.

Capítulo 8

Conclusões

A tese propôs um arcabouço completo para cadeias produtivas agropecuárias, incluindo: um modelo e uma arquitetura para cadeias produtivas agropecuárias, um processo de negociação e um formato para contratos multi-laterais que visam estabelecer acordos entre participantes da cadeia, uma estratégia de utilização do processo de negociação e do contrato multi-lateral para o estabelecimento de organizações virtuais e a implementação de um *middleware* para o suporte do processo de negociação baseado num motor de *workflow* (YAWL).

O arcabouço proposto contempla vários aspectos de uma cadeia produtiva, incluindo aspectos estruturais (quem são os atores e como se organizam) e sua dinâmica (como os atores interagem). O foco da tese concentrou-se no processo de negociação e no contrato entre os parceiros. Duas característica são relevantes para suas aplicabilidades: os contratos são multi-laterais e semi-formais. Como já mencionando, um conjunto de contratos bi-laterais não seria adequado para o contexto de cadeias produtivas agropecuárias. Também, contratos escritos em linguagens formais por especialistas seriam de difícil confecção. No outro extremo, contratos escritos em linguagem natural seriam dificilmente processados. Os outros aspectos serão abordados em trabalhos futuros.

Referências Bibliográficas

[1] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[2] W.M.P. van der Aalst, L. Aldred, M. Dumas, and A.H.M. ter Hofstede. Design and Implementation of the YAWL System. In A. Persson and J. Stirna, editors, *Advanced Information Systems Engineering, Proceedings of the 16th International Conference on Advanced Information Systems Engineering (CAiSE'04)*, volume 3084 of *Lecture Notes in Computer Science*, pages 142–159. Springer-Verlag, Berlin, 2004.

[3] W.M.P. van der Aalst, A.P. Barros, A.H.M. ter Hofstede, and B. Kiepuszewski. Advanced workflow patterns. In *CooplS '00: Proceedings of the 7th International Conference on Cooperative Information Systems*, pages 18–29, London, UK, 2000. Springer-Verlag.

[4] 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.

[5] W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.

[6] W.M.P. van der Aalst, P. Massuthe, C. Stahl, and K. Wolf. Multiparty Contracts: Agreeing and Implementing Interorganizational Processes. Technical report, Humboldt-Universität zu Berlin, 2007. Informatik-Berichte 213.

[7] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[8] 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.

[9] A Albani, A Keiblinge, K Turowski, and C Winnewisser. Identification and modelling of web services for inter-enterprise collaboration exemplified for the domain of strategic supply chain development. In R. et al. Meersman, editor, *CoopIS/DOA/ODBASE 2003*, pages 74–92, 2003.

[10] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, M. Montali, and P. Torroni. Expressing and verifying business contracts with abductive logic programming. *Intl. Journal of Electronic Commerce*, 12(4):9–38, summer 2008.

[11] 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.

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

[13] Arshinder, A. Kandan, and S.G. Deshmukh. A framework for evaluation of coordination by contracts: A case of two-level supply chains. *Computer & Industrial Engineering*, 56(4):1177–1191, 2009.

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

[15] E. Bacarin, W.M.P van der Aalst, E. Madeira, and C.B. Medeiros. Towards modeling and simulating a multi-party negotiation protocol with colored petri nets. In *Proc. CPN 07 - Eighth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, 2007.

[16] E. Bacarin, E.R.M. Madeira, and C.B. Medeiros. Contract e-negotiation in agricultural supply chains. *Intl. Journal of Electronic Commerce*, 12(4):71–97, summer 2008.

[17] E. Bacarin, E.R.M. Madeira, and C.B. Medeiros. Assembling and managing virtual organizations out of multi-party contracts. In J. Filipe and J. Cordeiro, editors, *ICEIS*, volume 24 of *Lecture Notes in Business Information Processing*, pages 758–769. Springer, 2009.

[18] E. Bacarin, E.R.M. Madeira, C.B. Medeiros, and W.M.P. van der Aalst. Spica's multi-party negotiation protocol: Implementation using yawl. *Intl. J. of Coop. Info. Sys.*, 2009. submitted.

[19] E. Bacarin, E.R.M. Madeira, and C.M.B. Medeiros. Using choreography to support collaboration in agricultural supply chains. Technical Report IC-07-07, Institute of Computing/UNICAMP, March 2007.

[20] E. Bacarin, E.R.M. Madeira, and C.M.B. Medeiros. Spica's multi-party negotiation protocol: Implementation using yawl. Technical Report IC-09, Institute of Computing/UNICAMP, March 2009.

[21] 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.

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

[23] P. Belfiore and H.T.Y. Yoshizaki. Scatter search for a real-life heterogeneous fleet vehicle routing problem with time windows and split deliveries in brazil. *European Journal of Operational Research*, 199(3):750–758, 2009.

[24] 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.

[25] B. Benatallah, M. Dumas, M.C. Fauvet, F.A. Rahbi, and Q.Z. Sheng. Overview of some patterns for architecting and managing composite web services. *ACM SIGecom Exchange*, 3(3):9–16, August 2002.

[26] B.L. Bhur. Information technology and changing supply chain behavior: Discussion. *Amer. J. Agr. Econ.*, 82(5):1130–1132, 2000.

[27] C. Bussler, D. Fensel, and A. Maedche. A Conceptual Architecture for Semantic Web enabled Web Services. *ACM Sigmod Record*, 31(4):24–30, 2002.

[28] K.E. Caggiano, P.L. Jackson, J.A. Muckstadt, and J.A. Rappold. Efficient computation of time-based customer service levels in a multi-item, multi-echelon supply chain: A practical approach for inventory optimization. *European Journal of Operational Research*, 199(3):744–749, 2009.

[29] A.M.G. Castro, S.M.V. Lima, W.J. Goedert, A. Freitas Filho, and J.R.P. Vasconcelos, editors. *Cadeias Produtivas e Sistemas Naturais – Prospecção Tecnológica.* EMBRAPA/Serviço de Produção de Informação, Brasília, 1998. in Portuguese.

[30] M. Cavalcanti, M. Mattoso, M. Campos, F. Llirbat, and E. Simon. Sharing Scientific Models in Environmental Applications. In *Proc ACM Symposium Applied Computing - SAC*, 2002.

[31] 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.

[32] D.C. Chatfield, T.P. Harrison, and J.C. Hayya. Scml: An information framework to support supply chain modeling. *European Journal of Operational Research*, 196(2):651–660, 2009.

[33] 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.

[34] P. Cramton, Y. Shoham, and R. Steinberg. An overview of combinatorial auctions. *SIGecom Exch.*, 7(1):3–14, 2007.

[35] J. Dang and M.N. Huhns. Concurrent Multiple-Issue Negotiation for Internet-Based Services. *IEEE Internet Computing*, 10(6):42–49, 2006.

[36] S. Darko-Ampem, M. Katsoufi, and P. Giambiagi. Secure negotiation in virtual organizations. In *EDOCW '06: Proceedings of the 10th IEEE on International Enterprise Distributed Object Computing Conference Workshops*, pages 48–55, Washington, DC, USA, 2006. IEEE Computer Society.

[37] 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.

[38] M.A. Figueiredo. Managing the quality of products in a supply chain (gerenciamento de regras de qualidade de produtos em cadeias produtivas). Master's thesis, Instituto de Computação - Unicamp, May 2009.

[39] R. Fileto, L. Liu, C. Pu, E. Assad, and C. B. Medeiros. POESIA: An Ontological Workflow Approach for Composing Web Services in Agriculture. *VLDB Journal*, 12(3), 2003.

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

[41] A. Gal and D. Montesi. Inter-enterprise workflow management systems. In *Proc. 10th International Conference and Workshop on Database and Expert Systems Applications (DEXA '99)*, pages 623–627, 1999.

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

[43] 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.

[44] 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.

[45] P.W.P.J. Grefen, N. Mehandjiev, G. Kouvas, G. Weichhart, and R. Eshuis. Dynamic business network process management in instant virtual enterprises. *Computers in Industry*, 60(2):86–103, 2009.

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

[47] B.N. Grosof and T.C. Poon. SweetDeal: Representing Agent Contracts with Exceptions Using Semantic Web Rules, Ontologies, and Process Descriptions. *Intl. Journal of Electronic Commerce*, 8(4):61–97, 2004.

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

[49] J.E. Hanson and Z. Milosevic. Conversation-oriented protocols for contract negotiations. *EDOC*, 00:40–49, 2003.

[50] 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.

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

[52] A.H.M. ter Hofstede, W.M.P. van der Aalst, M. Adams, and N. Russell. *Modern Business Process Automation: YAWL and its Support Environment*. Springer-Verlag, Berlin, 2010.

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

[54] E.M. Jewkes and A.S. Alfa. A queueing model of delayed product differentiation. *European Journal of Operational Research*, 199(3):734–743, 2009.

[55] O. Kallel, I.B. Jaâfar, L. Dupont, and K. Ghédira. Multi-agent negotiation in a supply chain - case of the wholsale price contract. In J. Cordeiro and J. Filipe, editors, *ICEIS (4)*, pages 305–314, 2008.

[56] A.A. Kondo. Gerenciamento de rastreabilidade em cadeias produtivas agropecuárias. Master's thesis, Instituto de Computação - Unicamp, Abril 2007.

[57] 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.

[58] 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.

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

[60] 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.

[61] 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.

[62] B. Medjahed, B. Benatallah, A. Bouguettaya, A.H.H. Ngu, and A.K. Elmagarmid. Business-to-business interactions: issues and enabling technologies. *The VLDB Journal*, 12(1):59–85, 2003.

[63] Z. Milosevic, S. Gibson, P.F. Linington, J. Cole, and S. Kulkarni. On design and implementation of a contract monitoring facility. In Boualem Benatallah, Claude Godart, and Ming-Chien Shan, editors, *Proceedings of WEC, First IEEE International Workshop on Electronic*, pages 62–70. IEEE Computer Society, July 2004.

[64] Z. Milosevic, A. Jøsang, T. Dimitrakos, and M.A. Patton. Discretionary enforcement of electronic contracts. In *EDOC*, pages 39–50. IEEE Computer Society, 2002.

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

[66] N.A. Mulyar. *Patterns for Process-Aware Information Systems: An Approach Based on Colored Petri Nets*. PhD thesis, Technische Universiteit Eindhoven, 2009.

[67] A.M. Nakai. Uma infra-estrutura para coordenação de atividades em cadeias produtivas baseada em coreografia de serviços web. Master's thesis, Instituto de Computação - Unicamp, Março 2007.

[68] 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.

[69] N. Oren, T.J. Norman, and A.D. Preece. Argumentation based contract monitoring in uncertain domains. In Manuela M. Veloso, editor, *IJCAI*, pages 1434–1439, 2007.

[70] T. Oszu and P. Valduriez. *Principles of Distributed Database Systems.* Prentice Hall, 1991.

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

[72] S. Panagiotidi, J. Vázquez-Salceda, S. Álvarez Napagao, S. Ortega-Martorell, , S. Willmott, R. Confalonieri, and P. Storms. Intelligent contracting agents language. In *Proceedings of the Symposium on Behaviour Regulation in Multi-Agent Systems -BRMAS'08. Aberdeen, UK*, pages 49–54, April 2008.

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

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

[75] J. V. Pitt, L. Kamara, M.J. Sergot, and A. Artikis. Formalization of a voting protocol for virtual organizations. In F. Dignum, V. Dignum, S. Koenig, S. Kraus, M.P. Singh, and M. Wooldridge, editors, *AAMAS*, pages 373–380. ACM, 2005.

[76] James A. Rappold and Nikolay Tchernev. Special issue on supply chain design. *European Journal of Operational Research*, 199(3):732–733, 2009.

[77] 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.

[78] N. Russell, W.M.P.van der Aalst, A.H.M. ter Hofstede, and D. Edmond. Workflow Resource Patterns: Identification, Representation and Tool Support. In O. Pastor and J. Falcao e Cunha, editors, *Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05)*, volume 3520 of *Lecture Notes in Computer Science*, pages 216—-232. Springer-Verlag, Berlin, 2005.

[79] N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Workflow Data Patterns: Identification, Representation and Tool Support. In L. Delcambre, C. Kop, H.C. Mayr, J. Mylopoulos, and O. Pastor, editors, *24nd International Conference on Conceptual Modeling (ER 2005)*, volume 3716 of *Lecture Notes in Computer Science*, pages 353–368. Springer-Verlag, Berlin, 2005.

[80] R. Rust and P. Kannan. E-Service: a New Paradigm for Business in the Electronic Environment. *Communications of the ACM*, 46(6):36–42, 2003.

[81] C.P. Sá, F.G. Andrade, and N.F. Almeida. Estudo da Cadeia Produtiva da Mandioca no Acre. In A.M.G Castro, S.M.V Lima, W.J. Goedert, A.F. Filho, and J.R.P. Vasconcelos, editors, *Cadeias Produtivas e Sistemas Naturais: Prospecção Tecnológica*, pages 321–341. Embrapa-SPI, 1998. in Portuguese.

[82] V. Salin. Information technology and cattle-beef supply chains. *American J. Agr. Econ.*, 82(5):1105–1111, 2000.

[83] Tuomas Sandholm and Victor Lesser. Leveled-commitment contracting: a backtracking instrument for multiagent systems. *AI Mag.*, 23(3):89–100, 2002.

[84] S.J. Simon. The art of military logistics – moving to dynamic supply chain. *Communications of the ACM*, 44(6):62–66, jun 2001.

[85] Y.B. Udupi and M.P. Singh. Contract enactment in virtual organizations: A commitment-based approach. In *AAAI*. AAAI Press, 2006.

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

[87] M. Weske, G. Vossen, C. B. Medeiros, and F. Pires. Workflow Management in Geoprocessing Applications. In *Proc. 6th ACM International Symposium Geographic Information Systems – ACMGIS98*, pages 88–93, 1998.

[88] L. Xu. A multi-party contract model. *SIGecom Exch.*, 5(1):13–23, 2004.

[89] R.S. Yamaoka, J.K. Watanabe, and S.A. Baroni. Estudo da cadeia produtiva da seda no estado do paraná. In A.M.G Castro, S.M.V Lima, W.J. Goedert, A.F. Filho, and J.R.P. Vasconcelos, editors, *Cadeias Produtivas e Sistemas Naturais: Prospecção Tecnológica*, pages 185–211. Embrapa-SPI, 1998. in Portuguese.

[90] E. Yücel, F. Karaesmen, F.S. Salman, and M. Türkay. Optimizing product assortment under customer-driven demand substitution. *European Journal of Operational Research*, 199(3):759–768, 2009.

[91] M. Zuzek, M. Talik, T. Swierczynski, C. Wisniewski, B. Kryza, L. Dutka, and J. Kitowski. Formal model for contract negotiation in knowledge-based virtual organizations. In *ICCS 2008, Part III, LNCS 5103*, pages 409–418, Berlin, 2008. Springer-Verlag.