Automatic Service Agreement Negotiators in Open Commerce Environments

Manuel Resinas, Pablo Fernández, and Rafael Corchuelo

ABSTRACT: There is a steady shift in e-commerce from goods to services that must be provisioned according to service agreements. This study focuses on software frameworks to develop automated negotiators in open commerce environments. Analysis of the literature on automated negotiation and typical case studies led to a catalog of 16 objective requirements and a conceptual model that was used to compare 11 state-of-the-art software frameworks. None of them was well suited for negotiating service agreements in open commerce environments. This motivated work on a reference architecture that provides the foundations to develop negotiation systems that address the previous requirements. A software framework was devised to validate the proposal by means of case studies. The study contributes to the fields of requirements engineering and software design, and is expected to support future efforts of practitioners and researchers because its findings bridge the gap among the existing automated negotiation techniques and lay the foundations for developing new software frameworks.

KEY WORDS AND PHRASES: Automated negotiation, electronic services, negotiation requirements, reference architecture, service agreements, services science, software frameworks.

E-commerce has evolved from shopping for goods on the Internet to outsourcing electronic services, such as flight reservations, payments, or executing business intelligence jobs on the cloud [10, 47, 56]. The focus of the present study is on so-called next-generation companies, which rely heavily on the automation of business processes that build on services outsourced around the world because this enables companies to be more efficient and cost-effective by exploiting economies of scale [4, 12, 13, 14, 15, 22, 40, 55]. For instance, Gartner predicts that at least 30 percent of the investment in software will go to outsourced services instead of product licenses by 2012, and 40 percent of capital expenditures will be made for outsourced infrastructure by 2011 [44].

Services science focuses on merging results in computer science, software engineering, and classical business sciences so that business requirements are better mapped onto technology [5]. The complex issues involved in this mapping require the simultaneous development both of business methods and of the technology that supports them. Fortunately, the technologies of service-oriented architecture (SOA) seem to be helping business analysts

This work was funded in part by the European Commission (FEDER—El Fondo Europeo de Desarrollo Regional), the Spanish Ministry of Science and Innovation, and the Andalusian government. The work by Manuel Resinas and Pablo Fernández was supported by grants TIN2006–00472 (Web-Factories), TIN2009–07366 (SETI), and P07-TIC-2533 (Isabel); the work by Rafael Corchuelo was supported by grants TIN2007–64119, P07-TIC-02602, P08-TIC-4100, and TIN2008–04718-E (IntegraWeb).

and software architects bridge the gap between the two worlds seamlessly [14, 42]. According to Oracle, companies that implement an SOA are able to reduce the costs of the integration and maintenance of projects by at least 30 percent [58].

However, this is not enough: It is also necessary to be effective, efficient, and flexible in the fast-changing conditions of the current market. Outsourcing is expected to evolve into dynamic outsourcing, whereby automated business processes can search, assess, and select the appropriate IT service providers and levels on demand [13, 26]. For dynamic outsourcing to become a reality, a number of critical issues must be addressed—for example, semantic discovery, service agreement negotiation, interoperability, monitoring, management, and governance, to name a few.

In the field of service agreement negotiation, the development of protocols, decision-making algorithms, and preferences and agreement models with desirable characteristics pose important challenges that have stimulated sig-nificant research efforts by the automated negotiation research community. However, the successful development of automated negotiation systems re-quires an understanding of the whole system as well as of the requirements of the negotiation context so that the most appropriate negotiation protocol, decision-making algorithms, and model are selected and put together in an automated negotiation system [29].

This article takes a software engineering approach in dealing with the latter problem. The ultimate goal is to understand the requirements of automated negotiation systems used to make service agreements in open commerce environments and to provide the foundations for developing such systems. The contributions in this article focus on two areas of the current software engineering body of knowledge [1].

The first contribution is to the field of requirements engineering. An exhaus-tive review of the literature together with analysis of several case studies led to the identification of several key requirements in the context of automated negotiation of service agreements in open commerce environments. Note that according to Bleistein et al., little attention has been given to compiling a catalog of requirements for e-business applications [9]. The present effort, then, can be seen as a materialization of such a catalog for service agreement negotiation systems. The key requirements are related to the expressiveness of service agreements, the heterogeneity of the parties involved in a negotiation, the lack of complete information about them, and the implicit dynamism of open commerce environments. Building on these factors, a conceptual model was developed based on 16 objective criteria that make it possible to compare current and future proposals. This analysis leads to the conclusion that the negotiation context, which depends on the user of the automated negotiation system and the parties with which the system negotiates, is likely to change in open commerce environments. Consequently, automated negotiation systems for such environments should be able to accommodate all these changes without requiring a great deal of effort.

The second contribution pertains to software design. Since there is an important need to embrace change, the next logical step is to analyze the current literature on software frameworks to develop automated negotiation systems. The conclusion is that neither protocol-oriented frameworks nor

intelligence-oriented frameworks address the aforementioned key requirements simultaneously, which shows the conceptual complexity of putting together automated negotiation techniques in a unique framework [3, 6, 7, 24, 30, 32, 35, 43, 45, 52, 54]. This motivated the effort to develop a reference architecture for automated negotiation systems. A reference architecture is not the description of a concrete system, but a reusable design that defines the tasks, grouped into roles, and the interactions that must be implemented in an automated negotiation system. It is a foundation for developing automated negotiation systems that facilitates their maintainability and their adaptability to changes in their context.

Finally, the third contribution is the software framework that has been developed. It demonstrates the soundness and usefulness of the reference architecture, and, as well, provides a reference implementation in which automated negotiation techniques (protocols, algorithms, and models) can be integrated. It was validated by means of several case studies.

These contributions can be used to support future efforts of practitioners and researchers in the automated negotiation field.

Practitioners can use the requirements and the reference architecture to set the basis on which new software frameworks for automated negotiation systems can be built. Note, too, that a number of protocols, algorithms, and models have already been integrated into an accompanying software toolkit. A practitioner for whom this toolkit is enough can deploy the framework immediately. Our implementation of the software framework may be an excellent starting point in developing advanced features, such as integration with other enterprise systems (e.g., enterprise service buses or business process management systems).

Researchers will obtain threefold support from the reference architecture. First, the requirements identified may help guide research efforts on new negotiation models that deal better with the problems identified regarding automated negotiations in open commerce environments. Second, the architecture may help researchers on automated negotiation to be aware of the tasks for which their negotiation model must account. More important, the architecture enables researchers to focus on a particular part of an automated negotiation system (e.g., world modeling algorithms or response generation algorithms). At the same time it bridges the gap between current automated negotiation techniques (protocols, algorithms, models), and thus eases the reuse of current results in this field. Finally, since there is an implemented software framework based on the reference architecture, a researcher who works on decision-making can integrate a proposal into the framework, test it, and compare the results. This enables comparison of several proposals from an empirical point of view, which has usually been a major drawback according to the literature surveyed.

Conceptual Model and Requirements for Automated Negotiation Systems

The discussion in this section details the conceptual model of an automated negotiation system, then analyzes the key problems a service agreement

negotiation system must face in open commerce environments, and reports on 16 requirements for automated negotiation systems to deal with these problems. Finally, it reports on the proposals surveyed and analyzes how well they support these requirements.

Conceptual Model

Figure 1 sketches a conceptual map regarding negotiation systems that derives from the survey of the literature and analysis of several case studies.

The *user* of an *automated negotiation system* defines *preferences,* namely, the data used to ensure that an agreement is reached according to the *user*'s needs. The user initiates the *automated negotiation system* to negotiate on his or her behalf. The *automated negotiation system* is also provided with *party references.* A *party reference* gives a means to interact with a *party.* The *user* may provide *party references* or they may come from a *party* as a request to start a negotiation.

For each *party reference* received, the *automated negotiation system* starts a process whose goal is to execute a *negotiation* with the *party* whose *reference* was received. *A negotiation* is a concrete execution of a *negotiation protocol* played by two or more *parties. Negotiations* are configured by a *protocol configuration,* which specifies characteristics of the *negotiation protocol* for a particular execution. For instance, they may specify the timeout of the protocol or its security features.

The execution of a *negotiation* involves the exchange of *negotiation messages,* whose specific characteristics are determined by a *negotiation protocol.* Each *negotiation message* has a *performative* to express the intention of the sender about the message and a *message content.* Depending on the *performative, negotiation messages* can be classified as *binding negotiation messages,* which involve a firm commitment with the other *party,* and *nonbinding negotiation messages,* which do not involve a firm commitment. *Nonbinding negotiation messages* can be used to give additional information to the other party or to explore new choices without committing to them. This is very useful in dynamic contexts in which there are several simultaneous negotiations with different *parties* and the feasibility of committing to an agreement depends on the current state of the service provider's resources.

The *message content* of a *negotiation message* is usually a *proposal.* However, other kinds of information can be exchanged, such as *threats, rewards,* or *arguments.* A *proposal* is an offer of an agreement made by one *party*. It specifies the *parties* to which it pertains and a set of *terms. Terms* designate both functional descriptions and nonfunctional guarantees of the service. Some examples of usual *terms* include: "the service interface is specified in the document that is available at http://example.org/myservice.wsdl," "the response time is less than 20 ms," or "the number of service requests is lower than 10 times per minute." Additionally, *proposals* may also include *negotiation data* to express additional information to guide the negotiation process.

When *parties* agree on a *proposal,* an *agreement* is created. An *agreement* is a document that defines a relationship between *parties.* Its goal is to define the *terms* that regulate the execution of a service, and it must have a specification

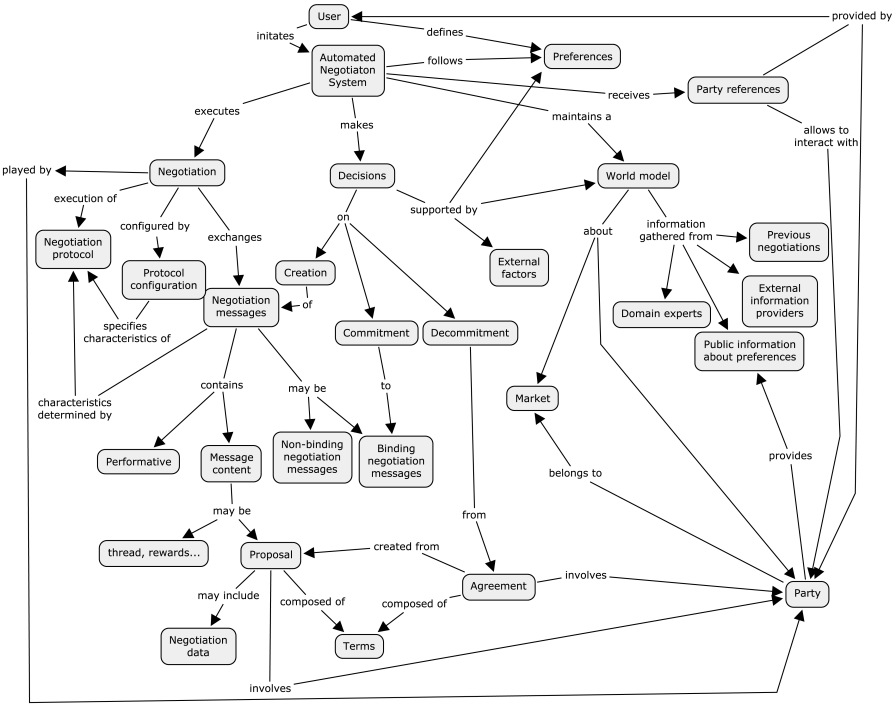

Figure 1. Conceptual Map of Automated Negotiation Systems

of the parties involved and a collection of *terms,* such as those described in the *proposal.* These *terms* regulate how the execution of the service must be carried out in the context of the agreement. In addition, unlike *proposals,* in which the *terms* can be left open in order to be refined later, *agreement terms* must be fully specified and ambiguities must be avoided.

The execution of a *negotiation* requires the *automated negotiation system* to make several *decisions* on the *creation* of *negotiation messages* (what *negotiation message* must be sent to the other party), the *commitment* to *binding negotiation messages* (whether the automated negotiation system must commit to an agreement and when it must do it), and the *decommitment* from previously created *agreements.* These decisions are supported by the information available to the automated negotiation system, which is the set of *preferences* sent by the *user,* a *world model* the automated negotiation system may maintain, and *external factors* that may have an influence on the decision, such as the provider's capacity to accept a new agreement.

Finally, the *world model* maintained by the *automated negotiation system* may comprise models created about the *market,* other negotiating *parties,* or the *negotiation domain.* To create these models, the *automated negotiation system* needs to gather information from several sources, which include *domain experts, public information about preferences* facilitated by the other *parties* themselves, *external information providers,* and by means of an analysis of *previous negotiations.*

Key Problems and Requirements

Expressiveness of Service Agreements

The negotiation of a service agreement usually involves such terms as availability, response time, security, and price. The negotiating parties are able to make trade-offs among the terms according to their preferences. Therefore, for an automated negotiation system to support expressive-enough service agreements, it should:

1.1. **Support multiterm negotiation protocols.** Not all negotiation protocols allow for multiterm service agreements. For instance, most auctioning protocols, except for multiattribute auctions, only support the negotiation of one term, usually the price [8, 28]. Bargaining protocols, which involve exchanging proposals and counterproposals among the parties, usually support multiterm negotiations.

1.2. **Manage expressive preferences.** To enable trade-offs, preferences must be expressed in a formalism that captures the relationships between terms, for example, utility functions, combinations of attributes, or fuzzy constraints [17, 19, 37].

Parties Are Heterogeneous

The best negotiation protocol, decision-making algorithm, preferences model, and agreement model depend on the negotiation context, which involves at least two stakeholders: the user of the automated negotiation systems and the parties with which the system negotiates [29]. The problem is that in open commerce environments, this negotiation context is likely to change. These changes derive from two sources.

First, in open commerce environments, new parties may appear unexpectedly, implement a variety of negotiation protocols, and have diverging behaviors during the negotiation. For example, some parties can concede more at the beginning of the negotiation, whereas others may concede only when the deadline is approaching and express preferences and agreements using different models.

Second, the requirements of the user of an automated negotiation system have a strong influence on the system because there is a trade-off between the expressiveness of the agreement and preferences models and the availability and complexity of the corresponding decision-making algorithms. Therefore, models and algorithms may change depending on the user's needs. The problem here is that the user's needs are subject to continuous adaptation and variation, adding new business rules and regulations, types of business-related events, operations, and so forth [41].

Automated negotiation systems in open commerce environments should be able to accommodate all these changes without requiring a great devel-

opment effort. Hence, it would be desirable for an automated negotiation system to:

2.1. Support multiple negotiation protocols. Since there is no standard negotiation protocol, different parties may implement different negotiation protocols. An automated negotiation system should support several negotiation protocols to avoid losing business opportunities.

2.2. Negotiate the negotiation protocol. Although all bargaining protocols involve the exchange of proposals between parties, the exchange may be carried out without restrictions on the content of the proposal and counterproposal or with restrictions on the order in which terms are negotiated or on the terms of the proposals [18, 20, 31]. It is desirable for negotiation systems to be able to choose the most suitable negotiation protocol for each context—depending, for example, on the number of terms under negotiation or the negotiation deadline.

2.3. Support multiple decision-making algorithms. There are a variety of decision-making algorithms based on game-theoretic approaches, heuristic approaches, and evolutionary approaches [18, 19, 21, 33, 37, 38]. Their effectiveness depends on the behavior of the other parties [29, 46]. An automated negotiation system should support several decision-making algorithms and choose the most appropriate one at run time.

2.4. Support multiple agreement models. Agreements can be expressed using formalisms that range from name-value pairs to ontologies or deontic logic [6, 18, 25]. Supporting multiple agreement models enables the user to make a trade-off between the expressiveness it requires and the availability and complexity of the corresponding decision-making algorithms.

2.5. Support multiple preferences models. There are many formalisms to express preferences: utility functions, constraints, fuzzy constraints, a combination of attributes, and rules [16, 17, 18, 23, 24, 33, 37]. Two issues must be considered before deciding which formalism is the most appropriate: the negotiation domain and the influence of the model on the decision-making algorithms with regard to availability and complexity.

2.6. Allow for user preferences about negotiation processes. Usually, not all parties have the same preferences about a negotiation process. Some parties may have a shorter deadline, may be eager to reach an agreement, or may be less strict about the agreements they accept. Furthermore, the user that sets the preferences may be a software system. For instance, a component may analyze the current state of the business processes and provide the automated negotiation system with the appropriate guidelines for the negotiation depending on the characteristics and the state of those processes (e.g., if a business task is on a slack path for a process

workflow). This enables the integration of the automated negotiation system with other parts of the IT infrastructure.

Partial Information About Parties

Having information about other parties strengthens one's negotiating capability or, complementarily, weakens others' capabilities. Unfortunately, automated negotiation systems do not usually have complete information about the parties with which they negotiate [11, 37, 59]. Therefore, it would be desirable for an automated negotiation system to:

- **3.1. Manage different types of knowledge about other parties.** This can be either knowledge about their preferences, knowledge about their behavior during the negotiation (e.g., whether they tend to concede, their negotiation deadline), and knowledge about the party itself (e.g., reputation or geographical location) [39, 59].
- **3.2. Gather information from different sources.** This includes information provided by a domain expert. Fatarin, Sierra, and Jennings propose that a domain expert must provide a measure of similarity between values of terms in the negotiation domain [19]. There is also information gathered directly from the other party. For example, in WS-Agreement, parties' templates can be used to learn what kind of agreements other parties are willing to accept [2]. Information may also be gathered from external information providers, such as reputation providers.
- **3.3. Build analysis-based models of parties.** Messages exchanged with other parties during previous negotiations can be analyzed to learn about their preferences and their behavior [11, 59]. This analysis can be classified into on-line analysis and off-line analysis, depending on whether the state of current negotiations is taken into account [34, 50, 59]. Some proposals use both kinds of analysis; for example, the one by Coehoorn and Jennings [11].

Markets Are Dynamic

The idleness of a resource results in a loss of revenue [24]. In consequence, providers commonly offer discount prices when their resources are likely to become idle. Several providers and consumers usually compete on the same services, which makes market conditions extremely volatile. To deal with these changing market conditions, an automated negotiation system should:

- **4.1. Support several negotiations simultaneously.** This is desirable because it would allow the system to choose the party that offers the most profitable agreement.
- **4.2. Select decision-making algorithms dynamically.** When dealing with simultaneous negotiations, the state of the negotiations can

have an influence on the decision-making algorithms used. For instance, if the system is carrying out several simultaneous negotiations and finds a very profitable agreement, it can take a tougher stance with the other parties. Therefore, it is desirable to be able to change the decision-making algorithm at run time, so that it can be adapted to changing contexts effectively [46].

4.3. **Support decommitment.** Decommitting from an agreement involves revoking it and paying a decommit fee [39, 48]. In a dynamic market, more profitable new offers may be found at any time. Hence, it is very convenient to be able to decommit from previous agreements. This topic is not sufficiently covered in the literature and requires further research before a complete framework can be developed.

4.4. **Supervised creation of agreements.** To avoid committing to agreements that cannot be satisfied, the automated negotiation system should be supervised by external elements, such as a capacity estimator to determine whether an agreement can be accepted or not [36].

4.5. **Build market models.** The characteristics of the market may have an influence on the negotiation process [50]. Therefore, it is convenient for an automated negotiation system to build models of the market to obtain information, such as the reservation price of a product or the chances that new parties will be found during a negotiation [34].

Analysis of Current Solutions

The key problems described in the previous section provide a number of objective requirements that can be used to compare current state-of-the-art automated negotiation frameworks. Table 1 and Table 2 summarize the comparison: a ✓ in a cell means that the corresponding proposal provides explicit support for the corresponding feature; a ~ indicates that it addresses it partially; an × indicates that the feature is not supported; NA means that there is no information available. (Note that proposals such as those of Kowalczyk and of Su et al. are not taken into account because they are specific-purpose negotiation systems, not software frameworks [33, 53]. They are not intended to be the foundation for building other negotiation systems, which is the focus in this article.)

Protocol-Oriented Frameworks

These frameworks provide the brawn of a negotiation system because they deal with the negotiation protocol and low-level interoperability issues.

Some of them define a negotiation host or marketplace that acts as a mediator among the negotiating parties. For instance, Kim and Segev describe a Web services–enabled marketplace architecture, which enables the automation

Table 1. Comparison of Automated Negotiation Frameworks (I).

Proposal	(1.1)	(1.2)	(2.1)	(2.2)	(2.3)	(2.4)	(2.5)	(2.6)
Kim and Segev [32]	✓	×	✓	×	×	×	×	×
Rinderle and Benyoucef [45]	✓	×	✓	×	×	×	×	×
Bartolini et al. [6]	✓	×	✓	×	×	×	×	×
Silkroad [52]	✓	×	✓	×	×	×	×	×
Ashri et al. [3]	✓	NA	✓	×	NA	NA	NA	NA
Ludwig et al. [35]	✓	✓	✓	×	✓	×	×	×
PANDA [24]	✓	✓	✓	×	✓	×	×	✓
DynamiCS [54]	✓	NA	✓	×	✓	×	NA	NA
Benyoucef and Verrons [7]	NA	NA	✓	×	✓	×	NA	NA
Jonker et al. [30]	✓	✓	NA	×	✓	×	×	✓
Paurobally et al. [43]	✓	✓	×	×	~	×	~	×

(1.1) Support multiterm negotiation protocols
(1.2) Manage expressive agreement preferences
(2.1) Multiple protocol support
(2.2) Negotiability of protocols
(2.3) Multiple decision-making algorithms
(2.4) Support multiple agreement models
(2.5) Support multiple preference models
(2.6) Allow user preferences about negotiation process

Table 2. Comparison of Automated Negotiation Frameworks (II).

Proposal	(3.1)	(3.2)	(3.3)	(4.1)	(4.2)	(4.3)	(4.4)	(4.5)
Kim and Segev [32]	×	×	×	×	×	×	×	×
Rinderle and Benyoucef [45]	×	×	×	×	×	×	×	×
Bartolini et al. [6]	×	×	×	×	×	×	×	×
Silkroad [52]	×	×	×	×	×	×	×	×
Ashri et al. [3]	✓	×	✓	×	×	×	×	×
Ludwig et al. [35]	×	×	×	×	×	×	×	×
PANDA [24]	✓	×	~	✓	×	×	✓	×
DynamiCS [54]	×	×	×	×	×	×	×	×
Benyoucef and Verrons [7]	NA	✓	×	×	✓	×	✓	×
Jonker et al. [30]	×	✓	✓	×	×	×	×	×
Paurobally et al. [43]	×	×	×	~	×	×	✓	×

(3.1) Manage different types of knowledge about parties
(3.2) Gather information from different sources
(3.3) Build analysis-based models
(4.1) Support several negotiations simultaneously
(4.2) Select decision-making algorithms dynamically
(4.3) Support decommitment
(4.4) Supervised creation of agreements
(4.5) Build market models

of B2B negotiations by defining executable negotiation protocols in Business Process Execution Language (BPEL) [32]. Rinderle and Benyoucef propose a service-oriented marketplace to manage negotiation protocols specified as state charts that are later mapped onto BPEL [45]. Similarly, Silkroad relies on a meta-model, the so-called roadmap, that is intended to capture the characteristics of a negotiation process, and an application framework, the so-called skeleton, that provides several modular and configurable negotiation service components [52].

Other authors focus on describing the modules required to manage negotiation protocols, not marketplaces. For instance, Bartolini, Preist, and Jennings present a taxonomy of rules that capture a variety of negotiation mechanisms and a simple interaction protocol based on FIPA (Foundation for Intelligent Agents) specifications that is used together with the rules to define negotiation protocols [6]. They also define a set of roles and an OWL (Web Ontology Language)–based language to express negotiation proposals and agreements [6].

Intelligence-Oriented Frameworks

These frameworks provide the brains of a negotiation system because they focus on decision-making and world-modeling. Note, however, that a few of them also deal with protocols, with an emphasis on decoupling them from the intelligence algorithms used [3, 24, 54].

Ashri, Rahwan, and Luck describe an agent-oriented architecture at a very high level of abstraction, which means that they leave unspecified such aspects as the preferences and agreement model [3]. Their architecture also lacks some advanced features, such as the negotiability of protocols, gathering information from different sources, and mechanisms to deal with dynamic markets.

Ludwig et al. present a framework for service agreement negotiation in service grids [35]. It builds on WS-Agreement and provides a protocol service provider and a decision-making service provider to deal with the negotiation process [2]. Unfortunately, it only supports one agreement and preference model, and it cannot deal with partial information and dynamic markets.

PANDA is a framework that mixes utility functions and rules to carry out the decision-making process [24]. The decision-making component relies on rules, utility functions, and an object pool with several estimation libraries, the negotiation history, and the current offer. It cannot deal with heterogeneous parties because it does not allow protocols to be negotiated or multiple preference and agreement models. Nor does it deal with partial information, and the object pool, an important element used to build analysis-based models, is not implemented, but only vaguely specified. Although this framework is appropriate for dynamic markets, it does not enable selection of decision-making algorithms at runtime; neither does it support decommitting from previous agreements or build market models.

DynamiCS is an actor-based framework developed by Tu et al. [54]. It makes a clear distinction between the negotiation protocol and the decision-making model, and it uses a plug-in mechanism to support new protocols and

strategies. It does not allow protocols to be negotiated or multiple agreement models. Furthermore, the preferences are unspecified, and it is not well suited to deal with partial information or cope with dynamic markets.

Benyoucef and Verrons present a framework based on the separation of protocols and strategies within a service-oriented architecture that facilitates deployment and integration with current infrastructures [7]. It does not provide data models for agreements or preferences; and it is unclear whether it supports multiterm negotiations or can manage different types of knowledge about parties. Another drawback is that the services that can be composed into the system are vaguely defined. In addition, it does not seem to be able to build models of parties or markets, and it does not provide any mechanisms to enable several concurrent negotiations or to decommit from an agreement.

Jonker, Robu, and Treur describe a component-based generic agent architecture for multiattribute negotiation [30]. Unlike other proposals, it provides advanced capabilities to deal with partial information about other parties, and it also copes with multiterm negotiations successfully. However, it has problems when negotiating with heterogeneous parties in dynamic environments because it only supports one model to express agreements and preferences. Whether it supports several negotiation protocols is unclear, and the framework does not support mechanisms to deal with dynamic markets.

Paurobally, Tamma, and Wooldridge describe a framework for Web service negotiation in grids [43]. It deals with the expressiveness of service agreements successfully but fails to cope with heterogeneous parties because it only supports a negotiation protocol and agreement model. Although it supports multiple preference models and intelligence algorithms, it seems that they are predefined and cannot be changed easily. It also has some problems when dealing with partial information. Regarding their support for dynamic markets, it allows agreements to be created in a supervised manner by means of the so-called WS-DAIOnt service. Although the framework seems to support several simultaneous negotiations, the authors do not delve into this issue.

Reference Architecture

The NegoFAST reference architecture provides the foundations for building general-purpose negotiation systems that address the requirements identified in the previous section. Note that a reference architecture is not the description of a concrete system, but a reusable design that defines the tasks automated negotiation systems must provide and how they are interrelated. In this sense, the reference architecture can be regarded as a conceptual framework that identifies the key tasks of an automated negotiation system rather than as the description of a specific automated negotiation system.

The design goal of NegoFAST is twofold. On the one hand, NegoFAST is designed to promote the reusability of its elements, so that they may be freely integrated as a whole to produce automated negotiation systems. On the other hand, NegoFAST is designed to be flexible enough to adapt to the models, algorithms, and protocols best suited for each negotiation scenario. To this end, NegoFAST is divided into a protocol-independent part called

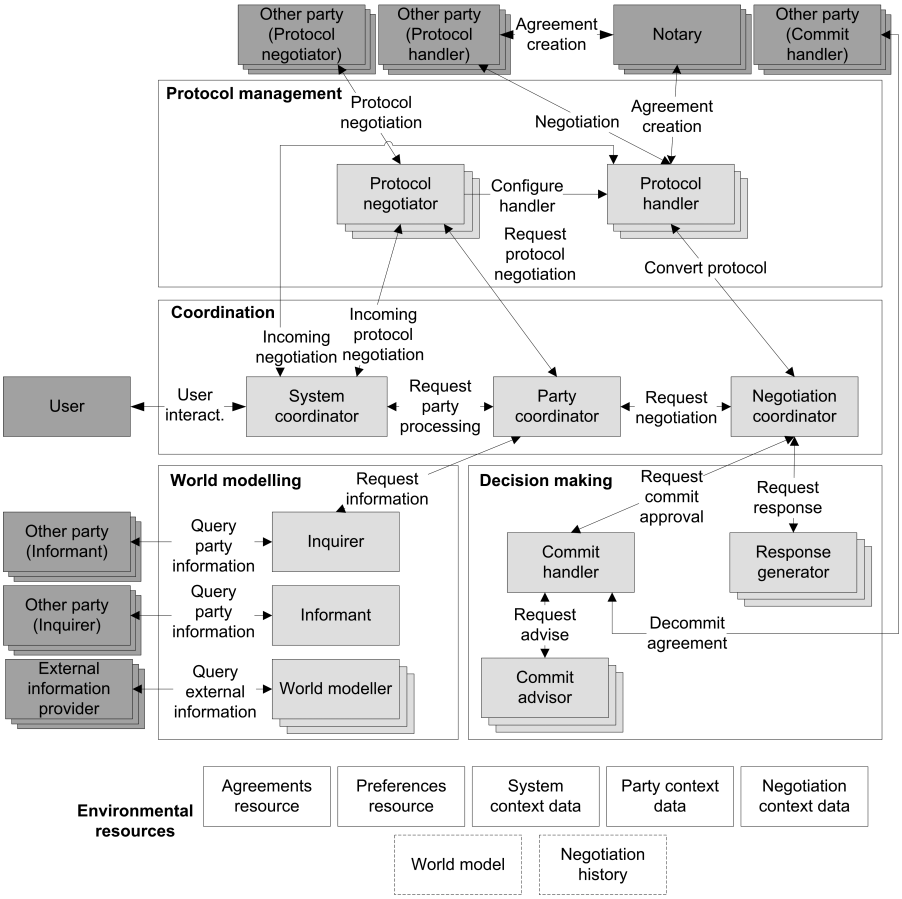

Figure 2. NegoFAST-Core Reference Architecture

NegoFAST-Core and protocol-specific extensions. Each extension deals with a group of similar negotiation protocols (e.g., extensions for bargaining protocols, auction protocols, or argumentation protocols). This article describes an extension for bargaining protocols (NegoFAST-Bargaining). NegoFAST also includes a data model that defines the data that are exchanged between the elements of the reference architecture.

NegoFAST-Core

Figure 2 depicts the NegoFAST-Core reference architecture. It is divided into four modules referred to as *Protocol Management*, *Coordination*, *World Modeling*, and *Decision-Making*. Each module is composed of several roles that define a group of related tasks, and they are depicted as light gray boxes. The arrows are interactions that represent exchanges of messages between the roles. The environment is divided into several resources that are depicted as white boxes.

The elements external to the architecture are depicted as dark gray boxes. Four of them are protocol-dependent and must be refined in protocol-specific extensions, namely: *ProtocolHandler, NegotiationCoordinator, ResponseGenerator,* and *NegotiationContextData.* The remaining elements are protocol-independent. A sequence diagram of a typical interaction between the roles in NegoFAST-Core is depicted in Figure 3.

Protocol Management Module

The Protocol Management module deals with the selection and execution of negotiation protocols. It is composed of two roles: *ProtocolNegotiator* and *ProtocolHandler.* A *ProtocolNegotiator* interacts with the other party to select and configure, if necessary, the negotiation protocol to be used. Consequently, it should implement an interaction protocol like SNego (REQ 2.2) [60].

A *ProtocolHandler* makes the automated negotiation system independent of the negotiation protocol selected by adapting the negotiation protocol into a generic negotiation protocol implemented by the *NegotiationCoordinator*. It also deals with communication errors that may occur during the negotiation and adapts messages to the NegoFAST data model. Adding support for a new negotiation protocol just involves implementing a new *ProtocolHandler* that adapts it to the generic negotiation protocol and the NegoFAST data model. If the negotiation protocol cannot be adapted to the generic negotiation protocol, a new *NegotiationCoordinator* must be implemented.

Note that a system may have several *ProtocolHandlers*. Thus a *ProtocolNegotiator* must choose among them in each negotiation (REQ 2.1).

Decision-Making Module

The Decision-Making module provides mechanisms to determine the behavior of the automated negotiation system during a negotiation. The module is composed of several *ResponseGenerators*, one *CommitHandler*, and several *CommitAdvisors*. Since the evaluation of a proposal is not a decision, it is not carried out by the decision-making module but by the *PreferencesResource.* A *ResponseGenerator* decides which negotiation messages must be sent. It is a protocol-specific role that must be detailed in protocol-specific extensions. There may be many *ResponseGenerators* in the same system to support multiple decision-making algorithms (REQ 2.3).

The *CommitHandler* decides when the system must commit to an agreement by sending a binding negotiation message. It also decides whether the system must decommit from an existing agreement (REQ 4.3). At run time, there must be just one *CommitHandler* to avoid concurrency problems with simultaneous negotiations (e.g., two agreements accepted at the same time). Optionally, the *CommitHandler* may use one or more *CommitAdvisors* to analyze the feasibility of accepting an agreement. They make their analysis based on domain-specific knowledge. For example, a *CommitAdvisor* may interact with

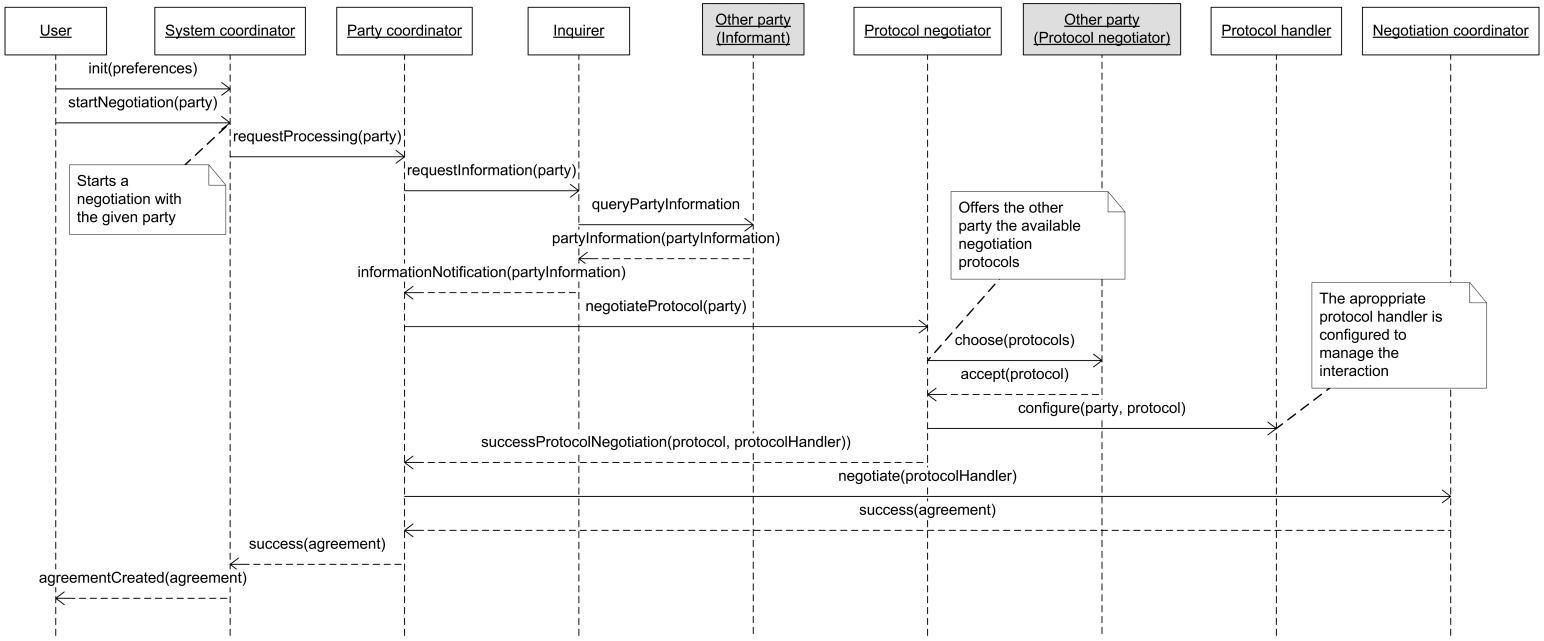

Figure 3. NegoFAST-Core Reference Architecture (sequence diagram)

the provider's resources to analyze whether it has enough capacity to provision a proposal (REQ 4.4).

The separation of the decision-making into these roles enhances the reusability of the different decision-making algorithms and makes it easier to adapt the system to changes in its negotiation context (REQ 2.3).

World Modeling Module

The World Modeling module gathers, analyzes, and manages useful information with which to make decisions during a negotiation. It is composed of an *Inquirer*, an *Informant*, and several *WorldModellers*. *Inquirer* and *Informant* enable polling the other parties to get information about them and the characteristics of the service demanded or offered. *Inquirer* gathers information from other parties by polling their *Informants*. The information gathered from other negotiating parties is stored in resource *PartyContextData*.

An automated negotiation system may have several *WorldModellers* that can be grouped into two categories: *WorldModellers* developed by domain experts that build models of the negotiation domain (e.g., a similarity measure between values of terms in the negotiation domain [19]), and *WorldModellers* that gather information from *ExternalInformationProviders* and messages exchanged in negotiations (REQ 3.2) and analyze them to build models of parties or the market (REQ 4.5) [11, 34, 46, 50, 59]. Note that it is important for market modelers to have access to *ExternalInformationProviders* to gather market information, such as querying a service registry to obtain information about the number of prospective providers (or competitors) in the market, or querying several auction sites to gather information about the results of recent auctions to provide measures, such as recommended maximum prices for an auction [27, 50].

To update their models, *WorldModellers* can perform off-line analysis of previous interactions using the *NegotiationHistory* or on-line analysis by means of a publish/subscribe mechanism provided by the environmental resources that notifies them when an event relevant for their models takes place (REQ 3.3).

Note that this design based on independent *WorldModellers* provides a flexible structure with which to manage different types of knowledge about parties easily (REQ 3.1), since each *WorldModeller* can focus on one type of knowledge about parties.

Coordination Module

The Coordination module coordinates the three levels of coordination contexts defined in NegoFAST-Core (system context, party context, negotiation context) by means of the three roles of which this module is composed.

SystemCoordinator coordinates the interactions between the system and the *User*, the initialization and termination of the system, and the reception of party references from the *User,* the *ProtocolNegotiator*, and the *ProtocolHandler*. The references are sent to the *PartyCoordinator* to be processed. It also updates the *SystemContextData*.

PartyCoordinator manages the processing of a party reference before the negotiation to get information about the party via an *Inquirer*, decide the negotiation protocol by means of a *ProtocolNegotiator*, and delegate the negotiation itself to a *NegotiationCoordinator*. It also updates the *PartyContextData*.

NegotiationCoordinator coordinates the execution of a negotiation by acting as a bridge between the *ProtocolHandler* and the *ResponseGenerator*. It invokes the *CommitHandler* when an approval to send a binding negotiation message is necessary. It stores the status of the negotiation in resource *NegotiationContextData*.

Environmental Resources

The environmental resources in the NegoFAST-Core reference architecture are data stores that can be read, modified, or both by the other roles. In addition, they provide a publish/subscribe mechanism to notify the roles when events take place. Environmental resources can be grouped into resources that are reinitialized in each execution of the system and resources that keep their information between different executions.

The former are the *AgreementsResource*, which stores all of the agreements made by the system so that they can be analyzed on the fly—for example, to decide whether it is convenient to decommit from an agreement. The *PreferencesResource* stores the users' preferences and allows the other roles to evaluate and compare agreements and proposals. *SystemContextData*, *PartyContextData*, and *NegotiationContextData* store information regarding the whole automated negotiation system (e.g., the time when the system was initialized, known party references), each negotiating party (e.g., the information gathered by the *Inquirer*, the negotiation protocol selected, the result of the negotiation), and each negotiation (e.g., its current state, the negotiation messages exchanged with the other parties).

The latter are the *WorldModel,* which stores the knowledge generated by the *WorldModellers,* and the *NegotiationHistory,* which allows it to build models based on previous interactions.

NegoFAST-Bargaining

The NegoFAST-Bargaining reference architecture extends NegoFAST-Core to deal with the specific requirements of concurrent bargaining negotiations (*see Figure 4*). A sequence diagram of a typical interaction between the roles in NegoFAST-Bargaining is depicted in Figure 5.

ProtocolHandler

The *ProtocolHandler* must adapt bargaining protocols into a generic negotiation protocol with the following characteristics: it is bilateral (i.e., there is only an initiator and a responder), it is sequential (the same party cannot send two negotiation messages in a row except for withdraw and cancellation messages),

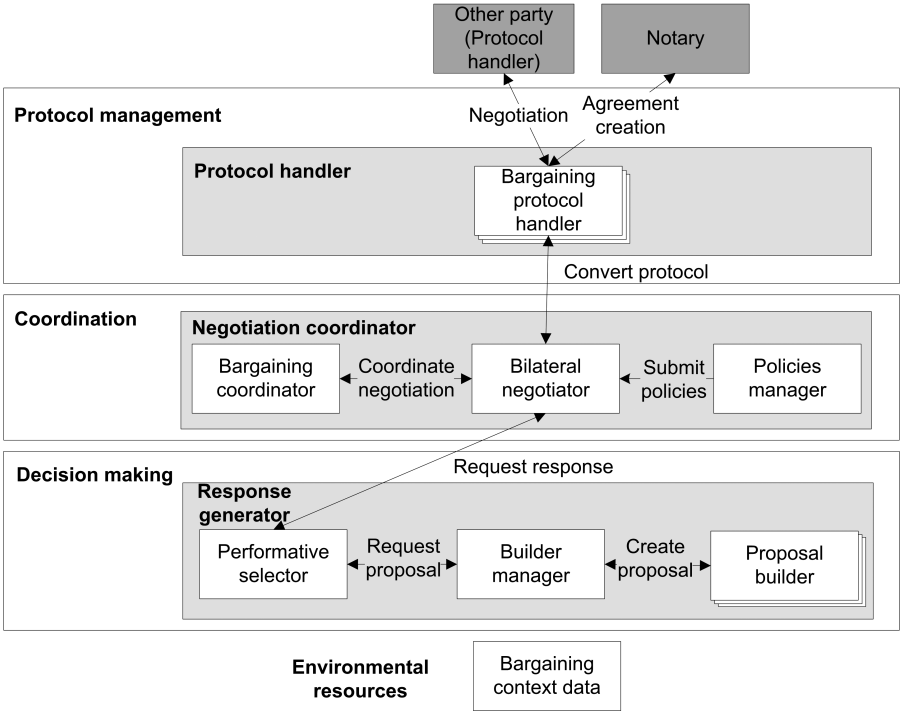

Figure 4. NegoFAST-Bargaining Reference Architecture

and it is proposal-based (the message content is composed of proposals only. In NegoFAST-Bargaining, the *ProtocolHandler* is refined into a specific-purpose role called *BargainingProtocolHandler*.

NegotiationCoordinator

The *NegotiationCoordinator* must be designed to support concurrent bilateral negotiations (REQ 4.1). To this end, it is refined into *BargainingCoordinator, BilateralNegotiator,* and *PoliciesManager. BargainingCoordinator* orchestrates the *BilateralNegotiator,* the *CommitHandler,* and the *PartyCoordinator,* and stores the current state of the concurrent negotiations in resource *BargainingContextData*.

BilateralNegotiator carries out a single bilateral negotiation by orchestrating the *BargainingProtocolHandler* and the *PerformativeSelector*. It communicates with the *BargainingCoordinator* to ask for approval before sending a binding negotiation message, and it receives negotiation policies from the *PoliciesManager*.

PoliciesManager uses the negotiation guidelines provided by the user's preferences together with the current state of the negotiations to determine specific negotiation policies that will guide the behavior of the *ResponseGenerator* (i.e., whether it should concede in the next proposal and how much it should

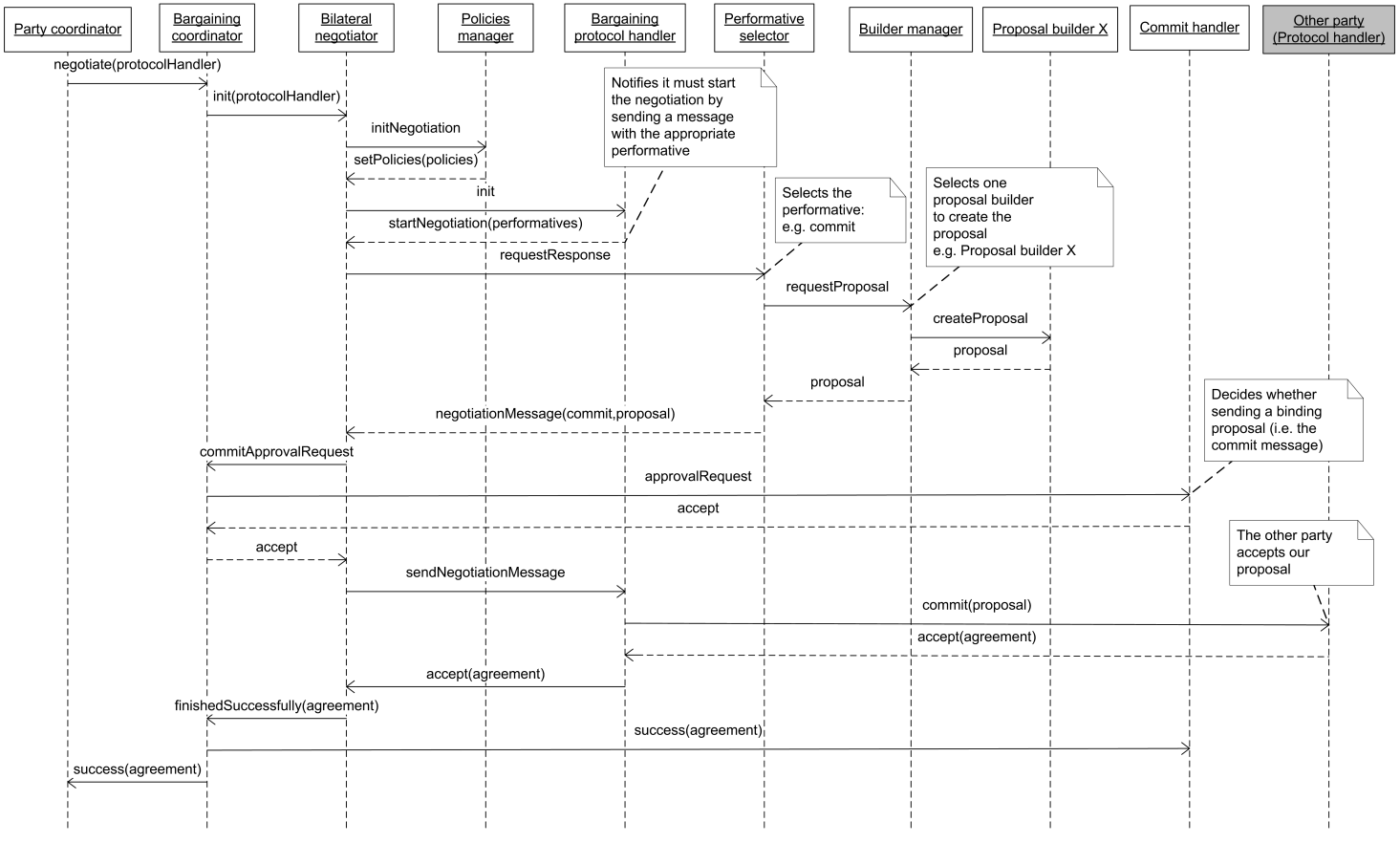

Figure 5. NegoFAST-Bargaining Reference Architecture (sequence diagram)

concede) during the negotiation and sends them to the *BilateralNegotiators*. By means of the negotiation policies, the *PoliciesManager* may guide the behavior of one negotiation based on how well the other negotiations are performing and, hence, properly support concurrent bilateral negotiations (REQ 4.1). For instance, if one negotiation is performing particularly well, (i.e., the proposals from the other party are very appealing), the negotiation policies of the other concurrent negotiations can be set to make the *ResponseGenerator* concede less. These negotiation policies are also used to ensure that the conduct of negotiations complies with the negotiation guidelines provided in the user's preferences (REQ 2.6).

Response Generator

Since negotiation messages are composed of a performative and a proposal, the *ResponseGenerator* can be refined into the *PerformativeSelector*, which selects the performative to be used, and the *BuilderManager* and *ProposalBuilders*, which create the accompanying proposal.

More specifically, the *BuilderManager* selects the most appropriate *ProposalBuilder* to create a new proposal. Each *ProposalBuilder* implements a decision-making algorithm that creates the proposals to be sent to the other party [18, 19, 33, 37]. Therefore, an automated negotiation system may have several *ProposalBuilders* that implement different decision-making algorithms (REQ 2.3), and the *BuilderManager* can choose dynamically which one should be used to create the proposal (REQ 4.2), for instance, using a technique like the one described by Ros and Sierra [46]. The *BuilderManager* can also configure the *ProposalBuilder* according to the aforementioned negotiation policies.

Negotiation Context Data

Environmental resource *NegotiationContextData* is extended by resource *BargainingContextData* to store and provide information about the status of current bargaining negotiations.

Data Model

The main contribution of the NegoFAST data model is that it defines a generic model that specifies the main concepts of an automated negotiation system (preferences, agreements, proposals, negotiation messages, and information about parties) independently of the formalism used to express them (*see Figure 6*). The concepts of the generic model are parametric. Therefore, creating a concrete model involves binding the parameters of the generic model to the concrete formalism used in the concrete model. Consequently, preferences and agreements can be as expressive as necessary, since any formalism (utility functions, rules, name-value pairs, constraints, fuzzy constraints, combinations

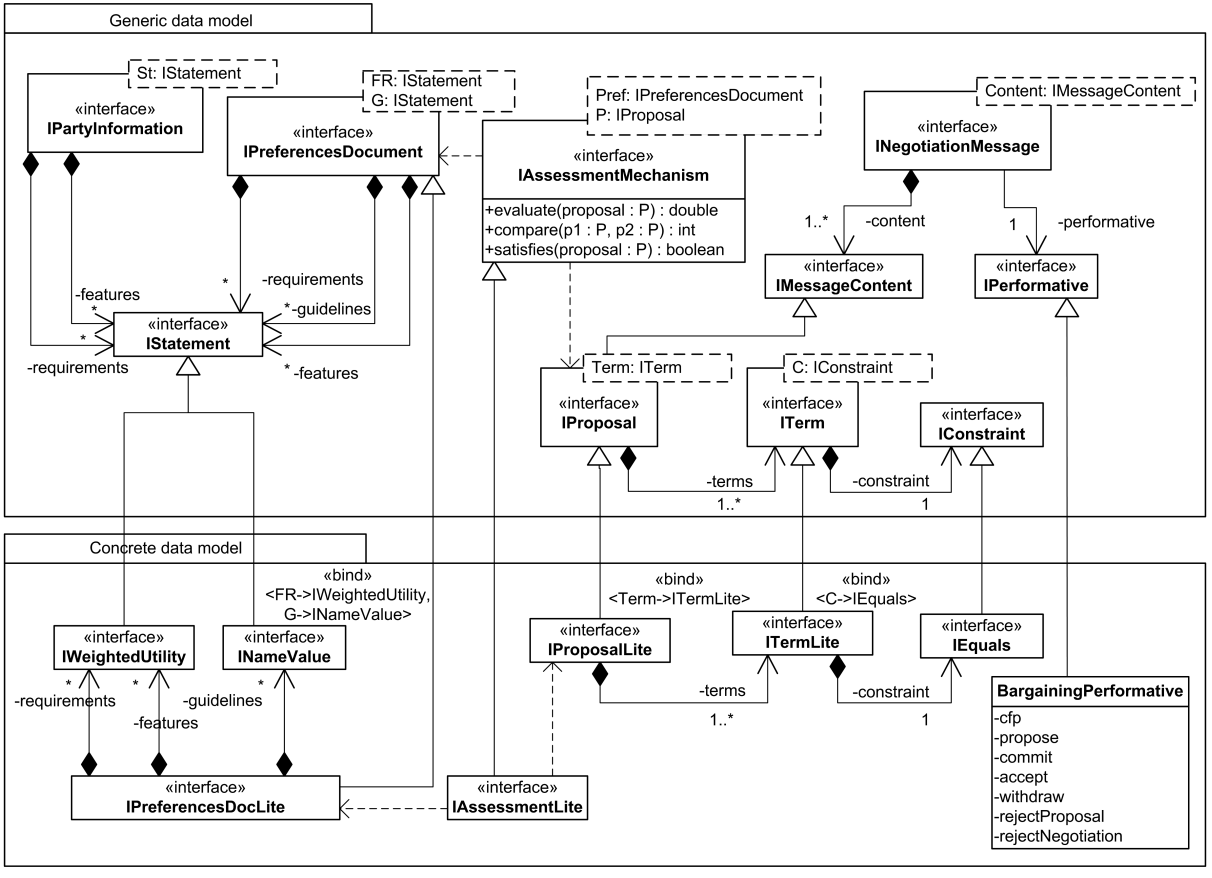

Figure 6. Preference, Agreement, and Proposal Models

of attributes) can be used provided that it complies with the following conditions (REQ 1.2):

- The formalism must extend the corresponding generic elements: IStatement for formalisms used to express preferences and information about parties, and ITerm for formalisms used to express agreements or proposals (*see Figure 6*).
- For each pair of preferences model and agreement model, an assessment mechanism (IAssessmentMechanism) must be defined that evaluates and compares two proposals that follow the given agreement model in the context of some preferences that follow the given preferences model.

The generic model itself can be extended to support advanced features. For instance, terms can be extended to add compensation clauses or proposals can be extended to include additional negotiation data about the terms specified in the proposal.

The main concepts of the generic data model are the following:

- **Preferences (interface IPreferencesDocument).** These are composed of three sets of statements (interface IStatement) about agreement-related features of the service to be provided, the requirements on other parties, and the negotiation guidelines that the automated negotiation system must follow. To allow different formalisms to express preferences, they are parameterized by the type of statements, such as utility functions, constraints, pair name-value, or rules (REQ 2.5). Note that the preferences about the negotiation process (the negotiation guidelines) are considered at the same level as the preferences about the service or about the other parties. This enables the definition of preferences that guide the behavior of the automated negotiation system, such as deadline, number of agreements to reach, and eagerness to reach an agreement (REQ 2.6).
- **Agreements and proposals (interfaces IAgreement and IProposal).** These are composed of a set of terms (REQ 1.1) and parameterized by the type of terms they contain. Terms (interface ITerm) specify constraints over some agreement-related features with which a party must comply and are parameterized by the type of constraint they enclose—for example, equality (see the concrete data model in Figure 6), constraints over one attribute, constraints over several attributes, or fuzzy constraints (REQ 2.4).
- **Negotiation messages (interface INegotiationMessage).** These consist of a performative—for example, propose, accept, or commit (interface Performative)—which must be refined by protocol-specific extensions (interface BargainingPerformative), and the contents of the message itself (interface IMessageContent), which is a tag interface that indicates which elements may be part of a negotiation message. Negotiation messages are parameterized by the type of their contents.

- **Information about parties (interface IPartyInformation).** This models the public information offered by the parties about their preferences and is obtained by means of role *Inquirer*. Like preferences, the party information is composed of two different sets of statements—requirements and features—and, also like preferences, it is parameterized by the type of statement used to express them. For instance, in Figure 6, a mix of weighted utility and name-value pairs is used.

Validation

To prove the soundness and usefulness of the NegoFAST reference architecture, the NegoFAST framework was designed and implemented to help validate the contributions by means of several case studies, namely:

- **Computing job outsourcing (Case 1).** This case study is developed in the context of a company that outsources computing power to run computing jobs (e.g., business intelligence jobs). Its goal is to implement an automated negotiation system for a computing job submitter that negotiates simultaneously with several job-hosting services to reach an agreement on the resources and cost required to execute one job.
- **Computing job-hosting service (Case 2).** This case study is similar to the previous one, but it focuses on the job-hosting service that negotiates with several computing job submitters.
- **Evolutive equilibrium (Case 3).** The goal of this case study is to show how to integrate the NegoFAST framework with a Java framework for genetic algorithms in order to apply a known evolutive approach to calculate the equilibrium among strategies.
- **Scheduling meetings (Case 4).** This case study focuses on the implementation of a mechanism to schedule meetings by means of the multiagent negotiations described by Wainer, Ferreira, and Constantino [57].

The goal of these case studies is threefold. First, implementing the case studies shows that the NegoFAST reference architecture can be translated into an implementation framework that can be used to effectively build automated negotiation systems.

Second, these case studies make it possible to check that both the reference architecture and the framework support the requirements described in the discussion of the background of automated negotiation systems. Table 3 summarizes the requirements covered by the different case studies. Note that Requirements 2.4 and 2.5 are covered by Case 1 and Case 4 together because each of them implements a different agreement and preferences model. (Although Requirements 3.2 and 4.5 have not been included in any of the case studies, the framework provides some specific market model algorithms [49].) Finally, no validation has been done regarding decommitment because the

Table 3. Requirements Covered by Case Studies.

Requirement	Case 1	Case 2	Case 3	Case 4
(1.1) Support multiterm negotiation protocols	✓	✓	✓	
(1.2) Manage expressive preferences models	✓	✓	✓	
(2.1) Support multiple protocols	✓			✓
(2.2) Negotiate negotiation protocol	✓	✓		
(2.3) Support multiple negotiation intelligent algorithms	✓	✓		✓
(2.4) Support multiple agreement models	✓			✓
(2.5) Support multiple preferences models	✓			✓
(2.6) Allow user preferences about negotiation process	✓	✓	✓	
(3.1) Manage different types of knowledge about parties		✓		
(3.2) Gather information from different sources				
(3.3) Build analysis-based models		✓		
(4.1) Support several simultaneous negotiations	✓	✓		
(4.2) Select intelligence algorithms dynamically	✓			
(4.3) Support decommitment				
(4.4) Supervised creation of agreements		✓		
(4.5) Build market models				

current version provides very little support for it. This topic is not sufficiently covered in the literature and requires further research before a complete framework can be developed.

Third, the careful selection of these four case studies makes it possible to check desirable nonfunctional properties of the framework: Case 2 (computing job-hosting service) was chosen to test the reusability of the framework because it is a similar scenario with differences regarding the decision-making roles; Case 3 (evolutive equilibrium) was chosen to test the ability of the framework to be integrated with other systems, an important feature in a realistic scenario; and Case 3 (evolutive equilibrium) and Case 4 (scheduling meetings) were chosen to test the adaptability of the framework to new scenarios.

These case studies were implemented in two stages. First, August, a proof-of-concept implementation of the NegoFAST framework, was developed using Java 1.5. August provides a reference implementation of the interfaces and generic data model specified in the NegoFAST framework. August was used to implement automated negotiation systems for the case studies.

Computing Job Outsourcing (Case 1)

This case study focuses on the negotiation of an agreement between a computing job submitter and several job-hosting services that need to agree on the job to be executed, the resources required, or scheduling requirements [2]. Figure 7 illustrates this scenario, and Figure 8 depicts its component diagram. First, the job submitter sends its preferences, which include both requirements about the job execution and guidelines regarding the negotiation process, to its

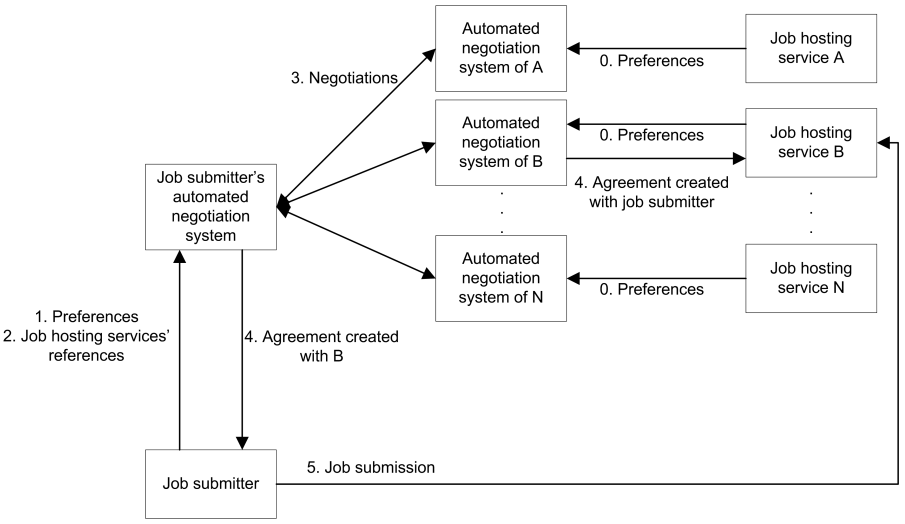

Figure 7. Computing Job Submission Scenario

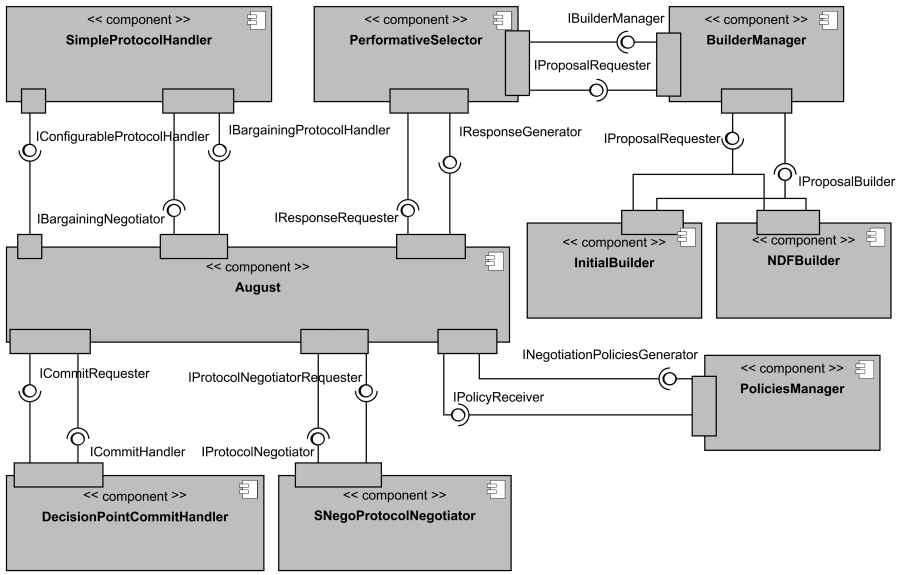

Figure 8. Automated Negotiation System for Computing Job Submission

automated negotiation system. In this case study, four guidelines are defined to control the negotiation process: negotiation deadline, number of agreements to reach, eagerness to reach an agreement, and minimum utility threshold. Second, when the automated negotiation system receives references to job-hosting services, it starts bilateral negotiations with them. When an agreement is reached, the automated negotiation system sends the agreement to the job

submitter. Finally, the job submitter sends the job to the job-hosting service, which executes it following the terms established in the agreement.

The automated negotiation system of this case study uses a common formalism to express preferences and agreements, which is the concrete data model depicted in Figure 6. Preferences are expressed as a mixture of weighted utility functions, which comprises an expressive preference model (REQ 1.2), (statements of type IWeightedUtility) for requirements and features, and name-value pairs (statements of type INameValue) for guidelines. Both agreements and proposals are expressed as equality constraints (interface IEquals). This is implemented by extending ITerm with interface ITermLite, which is parameterized by interface IEquals. An assessment mechanism (AssessmentLite) is implemented to evaluate the agreements and proposals with the weighted utility functions defined in the preferences.

The automated negotiation system supports the multiterm negotiation protocol described by Faratin et al. [18] (REQ 1.1), which is a bargaining negotiation protocol in which both parties exchange binding proposals until an agreement is reached or one party decides to finish the negotiation. This support is implemented in the *BargainingProtocolHandler*. In addition, a proof-of-concept *ProtocolNegotiator* is implemented based on the SNego protocol [60] (REQ 2.2). SNego is a protocol used to choose an option between a set of alternatives—in the present case, to choose a negotiation protocol from a set of alternative negotiation protocols.

Regarding the decision-making algorithm to create proposals, two algorithms are implemented (REQ 2.3). The first (InitialBuilder) creates a proposal by selecting the values that maximize the utility. The second (NDFBuilder) implements the decision-making algorithms proposed by Faratin et al. based on negotiation decision functions [18]. The dynamic selection of the decision-making algorithm is implemented in the *BuilderManager*, which selects InitialBuilder at the beginning of the negotiation. As the negotiation goes on, it starts selecting NDFBuilder (REQ 4.2). The *BuilderManager* is responsible for configuring the *ProposalBuilders* with the policies provided by the *PoliciesManager* based on the user's preferences (threshold, deadline, agreementsNumber, eagerness) (REQ 2.6).

Finally, the decision to commit to an agreement is made by *CommitHandler* based on decision points. When a decision point takes place, some binding negotiation messages are approved and the others are rejected. In the implementation, decision points take place when either the number of binding negotiation messages waiting for approval exceeds a certain threshold or the negotiation deadline is close. This *CommitHandler* also works as a coordination mechanism for the several simultaneous negotiations to ensure that the number of agreements reached complies with the preferences given by the user (REQ 4.1).

Computing Job-Hosting Service (Case 2)

To illustrate the reusability of the elements of the NegoFAST framework, the automated negotiation system described above was taken as a starting point

for analyzing the additional development that involves changing it to negotiate on behalf of the job-hosting service instead of the computing job submitter. In particular, the decision-making model was changed to a new one inspired by Nash's bargaining solution, which has no restrictions on the number of agreements that can be reached, provided that the hosting service has enough resources to execute it [38].

Since this decision-making model is based on the same formalism to express preferences and agreements as the one in the previous case study (REQ 1.2), the concrete model to express them can be reused. Similarly, the implementation of the *BargainingProtocolHandler* (REQ 1.1) and the *ProtocolNegotiator* (REQ 2.2) can be reused as well because the same negotiation protocols can be used in this case study. Finally, the implementation of the *PoliciesManager* can also be reused because the same policies are valid for the new decision-making model (REQ 2.6), except for policy agreementsNumber, which is ignored by the *CommitHandler*.

In contrast, new implementations of the decision-making roles must be provided. Namely, a new *ProposalBuilder* (class NashInspiredBuilder) to implement an algorithm inspired on the Nash bargaining solution that concedes in each proposal by trying to maximize the product of both utility functions; a new *BuilderManager* (class SimpleBuilderManager) to use the Nash-inspired builder instead of the negotiation decision functions builder; and a new *CommitHandler* (class ServerCommitHandler) so as not to impose any restrictions on the number of agreements that can be reached. Furthermore, this ServerCommitHandler uses a *CommitAdvisor* to evaluate whether the job-hosting service can accept an agreement given the currently available resources (REQ 4.4).

Finally, note that the NashInspiredBuilder requires building an analysis-based model (REQ 3.3) to estimate the utility function of the other party. This is implemented by means of a *WorldModeller* (UtilityFunctionEstimator) that uses the on-line learning mechanism described by Ros and Sierra to deduce the importance the other party gives to the negotiation terms based on the idea that the most important attributes for the other party are those with less variations between proposals [46].

Evolutive Equilibrium (Case 3)

Informally, a set of decision-making strategies are in equilibrium if each party has chosen a strategy and neither party can benefit by changing its strategy while the other players keep theirs unchanged. One approach to calculate equilibrium strategies is provided by the so-called evolutionarily stable strategies of Smith and Price [51]. In the present case study, the focus of interest is on building a system that applies a known evolutive approach to calculate the equilibrium among decision-making strategies based on negotiation decision functions [18, 21].

To this end, the NegoFAST framework is used to build a lightweight automated negotiation system that represents the individuals of both populations (consumer and provider) during the execution of the genetic algorithm and

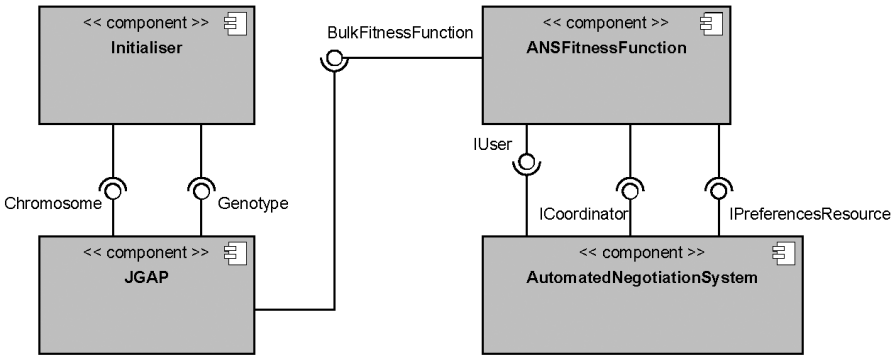

Figure 9. Integration of NegoFAST with JGAP

integrate it with JGAP, a Java framework to implement genetic algorithms (*see Figure 9*). The integration is carried out by developing a component (ANSFitnessFunction) that acts as a user of the NegoFAST framework. In each iteration of the algorithm, this component receives an individual's chromosome, which encodes the decision-making strategy, from the genetic algorithm system and maps it onto the preferences of each negotiator. After the negotiations, it computes the fitness function of each individual as the average utility obtained by the individual in the negotiations with the individuals of the other population. As iterations proceed, the decision-making strategies of all consumers and providers will converge to evolutionarily stable strategies.

Since a lightweight automated negotiation system that uses negotiation decision functions is needed, a simplified version of the automated negotiation system described in the first case study (REQ 1.1, REQ 1.2, REQ 2.6) is used. Specifically, the *ProtocolNegotiator* and the multiple decision-making algorithms are removed (i.e., just the NDFBuilder is retained), and the *CommitHandler* is simplified to commit to any agreement with a utility higher than the utility of the potential response. Finally, a new implementation of the *BuilderManager* is required to select the NDFBuilder in all cases.

Scheduling Meetings (Case 4)

This case study focuses on the implementation of a mechanism to schedule meetings by means of the multiagent negotiations described by Wainer et al. [57]. This case study can be used as an example of a significantly different formalism to express agreements and preferences, as well as a negotiation protocol (*see Figure 10*). Specifically, preferences and features are expressed as TimeSlotPreferences, a domain-specific preference in which a value is assigned to a specific time slot (REQ 2.5). Preferences guidelines are expressed as name-value pairs. However, in this case study, guidelines specify the type of negotiation the negotiator will carry out.

Proposals are expressed as a mixture of equality constraints to specify the length of a meeting and the beginning and ending of the time window in which

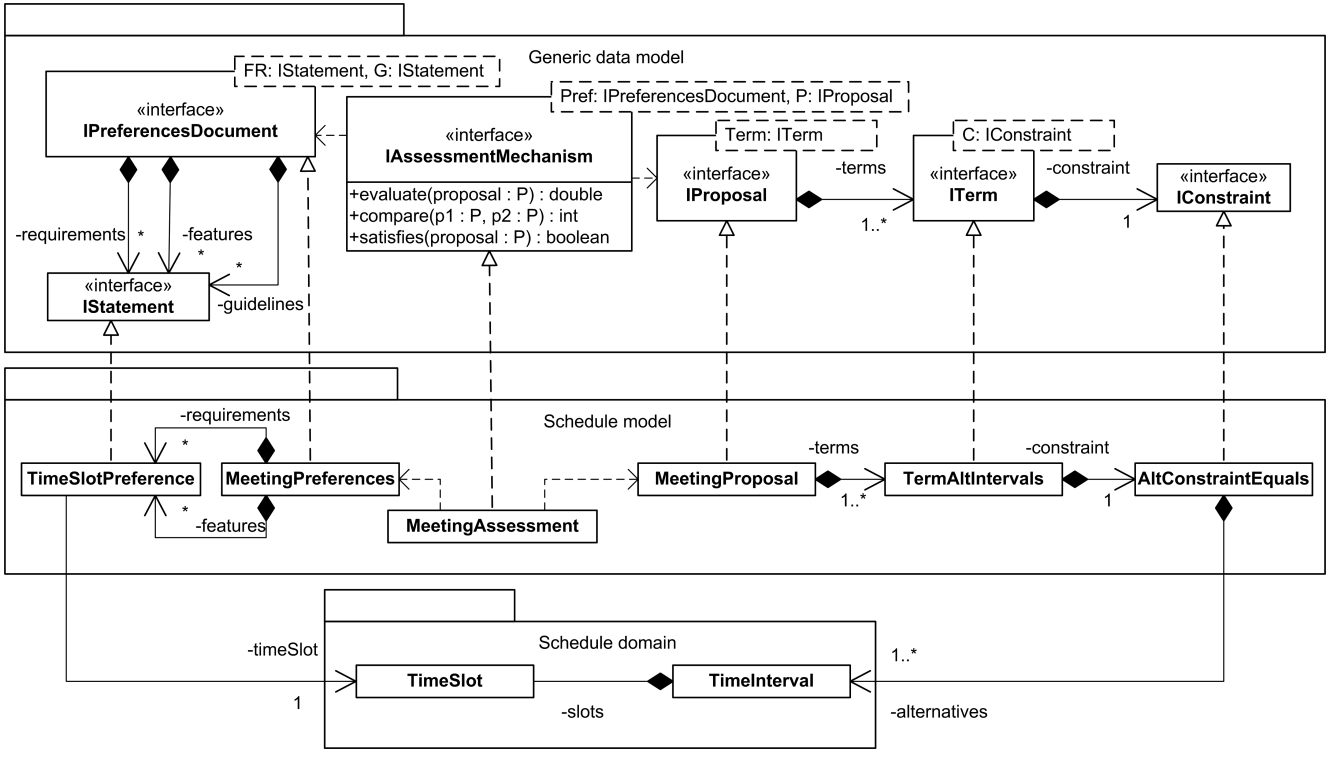

Figure 10. Preference and Proposal Models of Case 4 (scheduling meetings)

it must take place, and a domain-specific term to specify a set of alternative time intervals for the meeting (TermAltIntervals) (REQ 2.4).

MeetingProtocolHandler implements the negotiation protocol described by Wainer et al. [57] (REQ 2.1). In this case, the *ProtocolHandler* has to coordinate the submission and reception of proposals to and from participants.

Regarding the decision-making algorithm to create proposals, three algorithms are implemented (REQ 2.3) based on those described by Wainer et al. [57], namely: laconic, egotistic, and deceiving. The *BuilderManager* selects one of them based on the guidelines provided by the user's preferences. In addition, two *WorldModellers* have been developed to control the time slots sent to the other parties or suggested by them.

Conclusions

The present article focuses on the problem of building automated service agreement negotiation systems in open commerce environments. Services science brings together current work in computer science, software engineering, and more classical business-supporting sciences, such as operations research, management, or business strategy, to develop the skills required in a services-led economy. Not only is it important to be able to map business requirements onto IT capabilities, but it is also necessary to be effective, efficient, and flexible in the current market conditions, which are changing fast and are paving the way for dynamic outsourcing.

A number of key problems regarding dynamic outsourcing have been identified, namely: service agreement expressiveness, party heterogeneousness, partial information, and dynamic markets. From these problems, 16 well-founded requirements were elicited based on either an exhaustive review of the literature or case studies. These requirements were used as a conceptual model to analyze and assess current state-of-the-art proposals on automated negotiation frameworks. The conclusion is that none of the current protocol- or intelligence-oriented frameworks is complete with regard to the requirements. Furthermore, in open and dynamic environments, the negotiation context, which consists of the parties with which the system negotiates and the requirements of the user of the automated negotiation system, is likely to change. Consequently, automated negotiation systems should be able to accommodate all of these changes without requiring a great developmental effort. This requires that the automated negotiation system be carefully designed to ensure its maintainability and adaptability to changes in its context.

These conditions were the motivation for working on the NegoFAST reference architecture, which provides the foundations for building general-purpose negotiation systems that address these requirements. How NegoFAST supports these requirements is detailed above in the discussion of the reference architecture and is summarized in Table 4. Note that it not only integrates NegoFAST features from other frameworks, but it includes some unique features: the generic data model that enables the support of multiple preferences (REQ 2.4) and agreement models (REQ 2.5), the world model component, specially the design based on *WorldModellers* (REQ 3.3 and 4.5),

Table 4. How NegoFAST Deals with Requirements Identified.

Requirement	NegoFAST
(1.1) Support multiterm negotiation protocols	Agreement and proposal models with multiple terms
(1.2) Manage expressive preferences	Generic preferences model allows formalisms as expressive as necessary
(2.1) Support multiple negotiation protocols	Supported by different ProtocolHandlers and NegoFAST extensions
(2.2) Negotiate negotiation protocol	ProtocolNegotiator
(2.3) Support multiple negotiation decision-making algorithms	Different ProposalBuilders selected by BuilderManager
(2.4) Support multiple agreement models	Generic agreement model independent of formalism
(2.5) Support multiple preferences models	Generic agreement model independent of formalism
(2.6) Allow user preferences about negotiation process	Negotiation guidelines as first-level preferences and PoliciesManager to enforce them
(3.1) Manage different types of knowledge about parties	Independent WorldModellers that focus on one type of knowledge about parties
(3.2) Gather information from different sources	Use of Inquirer, Informant, and WorldModellers
(3.3) Build analysis-based models	WorldModellers that analyze environmental resources
(4.1) Support several negotiations simultaneously	BilateralNegotiators, BargainingCoordinator, and PoliciesManager
(4.2) Select decision-making algorithms dynamically	BuilderManager selects ProposalBuilders dynamically
(4.3) Support decommitment	Initial support in CommitHandler
(4.4) Supervised creation of agreements	CommitAdvisors provide domain-specific advice to CommitHandler
(4.5) Build market models	WorldModellers with access to ExternalInformation providers

and the use of a *ProtocolNegotiator* (REQ 2.2). Certain other features are less than common in current state-of-the-art negotiation frameworks, such as the ability to express user preferences about the negotiation process (REQ 2.6), support for several simultaneous negotiations (REQ 4.1), and the dynamic selection of decision-making algorithms (REQ 4.2). The only requirement that is not fully supported by NegoFAST is decommitment. In the current version of NegoFAST, the decommitment support is naive. The main reason is that decommitment is still a novel topic that deserves further research before being integrated into a framework.

On the basis of the NegoFAST reference architecture, a software framework has been designed and implemented, and it was used in this study to implement automated negotiation systems for several case studies that cover most of the aforementioned requirements. These case studies have also made it possible to check desirable nonfunctional properties of the framework, such as reusability, integrability, and adaptability.

The contributions made by this article will help practitioners since those contributions set the requirements and foundations to implement automated negotiation systems. They also will enable researchers to focus on a particular part of an automated negotiation system (e.g., world modeling algorithms or response generation algorithms), while the reference architecture bridges the gap between the current automated negotiation techniques (protocols, algorithms, models), and, hence, eases the reuse of current results in this field. The implementation of the software framework provides a harness in which automated negotiation techniques can be tested from an empirical point of view. For instance, researchers who work on decision-making algorithms to create agreement proposals to schedule meetings can proceed as follows: First, they can test their algorithms by integrating them as new *ProposalBuilders* in the system detailed in the discussion of scheduling meetings, without having to reimplement the whole automated negotiation system again. Second, they can compare their algorithms with others that have already been integrated into the framework (e.g., the laconic, egotistic, and deceiving algorithms). Third, they can analyze how well the algorithms perform in coordination with several different complementary models (e.g., how well they perform using different world-modeling algorithms).

REFERENCES

1. Abran, A.; Moore, J.W.; Bourque, P.; and Dupuis, R. *Guide to the Software Engineering Body of Knowledge.* Los Alamitos, CA: IEEE Computer Society Press, 2004.
2. Andrieux, A.; Czajkowski, K.; Dan, A.; Keahey, K.; Ludwig, H.; Nakata, T.; Pruyne, J.; Rofrano, J.; Tuecke, S.; and Xu, M. WS-Agreement recommendation. March 2007, www.gridforum.org/documents/GFD.107.pdf.
3. Ashri, R.; Rahwan, I.; and Luck, M. Architectures for negotiating agents. In V. Marík, J.P. Müller, and M. Pechoucek, M. (eds.), *Multi-Agent Systems and Applications III: 3rd International Central and Eastern European Conference on Multi-Agent Systems.* Heidelberg: Springer, 2003, pp. 136–146.

4. Bardhan, I.R.; Whitaker, J.; and Mithas, S. Antecedents of business process outsourcing in manufacturing plants. In R.H. Sprague (ed.), *39th Annual Hawaii International Conference on System Sciences.* Los Alamitos, CA: IEEE Computer Society, 2006, pp. 68–69.
5. Bardhan, I.; Demirkan, H.; Kannan, P.K.; Kauffman, R.J.; and Sougstad, R. An interdisciplinary perspective on IT services management and services science. *Journal of Management Information Systems, 26,* 4 (spring 2010), 13–65.
6. Bartolini, C.; Preist, C.; and Jennings, N.R. A software framework for automated negotiation. In R. Choren, A. García, C. Lucena, and A. Ramonovsky (eds.), *Software Engineering for Multi-Agent Systems III.* Berlin: Springer Verlag, 2005, pp. 213–235.
7. Benyoucef, M., and Verrons, M.-H. Configurable e-negotiation systems for large scale and transparent decision making. *Group Decision and Negotiation, 17,* 3 (May 2008), 211–224.
8. Bichler, M. An experimental analysis of multi-attribute auctions. *Decision Support Systems, 29,* 3 (October 2000), 249–268.
9. Bleistein, S.J.; Cox, K.; Verner, J.M.; and Phalp, K. Requirements engineering for e-business advantage. *Requirements Engineering Journal, 11,* 1 (December 2005), 4–16.
10. Christensen, C.M., and Raynor, M.E. *How to Avoid Commoditization.* Boston: Harvard Business School Press, 2003.
11. Coehoorn, R.M., and Jennings, N.R. Learning on opponents preferences to make effective multi-issue negotiation trade-offs. In M. Janssen, H.G. Sol, and R.W. Wagenaar (eds.), *Sixth International Conference on Electronic Commerce.* New York: ACM Press, October 2004, pp. 59–68.
12. Dai, Q., and Kauffman, R.J. Business models for Internet-based B2B electronic markets. *International Journal of Electronic Commerce, 6,* 4 (summer 2002), 41–73.
13. Dan, A.; Davis, D.; Kearney, R.; Keller, A.; King, R.P.; Kuebler, D.; Ludwig, H.; Polan, M.; Spreitzer, M.; and Youssef, A. Web services on demand: WSLA-driven automated management. *IBM Systems Journal, 43,* 1 (January 2004), 136–158.
14. Demirkan, H.; Kauffman, R.J.; Vayghan, J.A.; Fill, H.-G.; Karagiannis, D.; and Maglio, P.P. Service-oriented technology and management. *Electronic Commerce Research and Applications, 7,* 4 (winter 2008), 356–376.
15. Dobardziev, A. Outcome-based pricing: Incentivising innovation-led IT services. Ovum Knowledge Center, June 2008, http://store.ovum.com/Product.asp?pid=38842&etr=infaus.
16. Dujmovic, J.J. A method for evaluation and selection of complex hardware and software systems. In *22nd International Conference for the Resource Management and Performance Evaluation of Enterprise CS.* Turnersville, NJ: Computer Measurement Group, 1996, pp. 368–378.
17. Elfatatry, A., and Layzell, P.J. A negotiation description language. *Software, Practice and Experience, 35,* 4 (April 2005), 323–343.
18. Faratin, P.; Sierra, C.; and Jennings, N.R. Negotiation decision functions for autonomous agents. *International Journal of Robotics and Autonomous Systems, 24,* 3–4 (1998), 159–182.

19. Faratin, P.; Sierra, C.; and Jennings, N.R. Using similarity criteria to make trade-offs in automated negotiations. *Artificial Intelligence, 142,* 2 (December 2002), 205–237.
20. Fatima, S.S.; Wooldridge, M.; and Jennings, N.R. An agenda-based framework for multi-issue negotiation. *Artificial Intelligence, 152,* 1 (January 2004), 1–45.
21. Fatima, S.S.; Wooldridge, M.; and Jennings, N.R. A comparative study of game theoretic and evolutionary models of bargaining for software agents. *Artificial Intelligence Review, 23,* 2 (April 2005), 187–205.
22. Friedman, T. *The World Is Flat: A Brief History of the Twenty-first Century.* New York: Farrar, Straus, & Giroux, 2005.
23. Frølund, S., and Koistinen, J. Quality-of-service specification in distributed object systems. *Distributed Systems Engineering, 5,* 4 (December 1998), 179–202.
24. Gimpel, H.; Ludwig, H.; Dan, A.; and Kearney, B. PANDA: Specifying policies for automated negotiations of service contracts. In M.E. Orlowska, S. Weerawarana, M.P. Papazoglou, and J. Yang (eds.), *First International Conference on Service-Oriented Computing (ICSOC 2003).* Berlin: Springer-Verlag, 2003, pp. 287–302.
25. Governatori, G. Representing business contracts in RuleML. *International Journal Cooperative Information Systems, 14,* 2–3 (June 2005), 181–216.
26. Grefen, P.W.P.J.; Ludwig, H.; Dan, A.; and Angelov, S. An analysis of Web services support for dynamic business process outsourcing. *Information & Software Technology, 48,* 11 (November 2006), 1115–1134.
27. Gregg, D.G., and Walczak, S. Auction advisor: An agent-based online-auction decision support system. *Decision Support Systems, 41,* 2 (January 2006), 449–471.
28. He, M.; Jennings, N.R.; and Leung, H.-F. On agent-mediated electronic commerce. *IEEE Transactions on Knowledge and Data Engineering, 15,* 4 (July/August 2003), 985–1003.
29. Jennings, N.R.; Faratin, P.; Lomuscio, A.R.; Parsons, S.; Wooldridge, M.; and Sierra, C. Automated negotiation: Prospects, methods and challenges. *Group Decision and Negotiation, 10,* 2 (March 2001), 199–215.
30. Jonker, C.; Robu, V.; and Treur, J. An agent architecture for multi-attribute negotiation using incomplete preference information. *Autonomous Agents and Multi-Agent Systems, 15,* 2 (October 2007), 221–252.
31. Karp, A.H. Rules of engagement for automated negotiation. In B. Benatallah and C. Godart (eds.), *First IEEE International Workshop on Electronic Contracting.* Los Alamitos, CA: IEEE Computer Society, July 2004, pp. 32–39.
32. Kim, J.B., and Segev, A. A Web services-enabled marketplace architecture for negotiation process management. *Decision Support Systems, 40,* 1 (July 2005), 71–87.
33. Kowalczyk, R. Fuzzy e-negotiation agents. *Soft Computing, 6,* 5 (August 2002), 337–347.
34. Li, C.; Giampapa, J.; and Sycara, K. Bilateral negotiation decisions with uncertain dynamic outside options. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 36,* 1 (2006), 31–44.
35. Ludwig, A.; Braun, P.; Kowalczyk, R.; and Franczyk, B. A framework for automated negotiation of service level agreements in services grids.

In C. Bussler and A. Haller (eds.), *Business Process Management Workshops*. Berlin: Springer Verlag, 2005, pp. 89–101.
36. Ludwig, H.; Dan, A.; and Kearney, R. Cremona: An architecture and library for creation and monitoring of WS-Agreement. In M. Aiello, M. Aoyama, F. Curbera, and M.P. Papazoglou (eds.), *Second International Conference on Service-Oriented Computing (ICSOC 2004).* New York: ACM Press, November 2004, pp. 65–74.
37. Luo, X.; Jennings, N.R.; Shadbolt, N.; Leung, H.-F.; and Lee, J.H. A fuzzy constraint based model for bilateral, multi-issue negotiations in semi-competitive environments. *Artificial Intelligence, 148,* 1–2 (August 2003), 53–102.
38. Nash, J.F. The bargaining problem. *Econometrica, 18,* 2 (April 1950), 155–162.
39. Nguyen, T.D., and Jennings, N.R. Managing commitments in multiple concurrent negotiations. *Electronic Commerce Research and Applications, 4,* 4 (winter 2005), 362–376.
40. Ordanini, A. What drives market transactions in B2B exchanges? *Communications of the ACM, 49,* 4 (April 2006), 89–93.
41. Papazoglou, M.P. The challenges of service evolution. In Z. Bellahsene and M. Léonard (eds.), *20th International Conference on Advanced Information Systems Engineering (CAiSE 2008).* Berlin: Springer, June 2008, pp. 1–15.
42. Papazoglou, M.P., and van den Heuvel, W.-J. Service oriented architectures: Approaches, technologies and research issues. *International Journal on Very Large Data Bases, 16,* 3 (July 2007), 389–415.
43. Paurobally, S.; Tamma, V.; and Wooldridge, M. A framework for Web service negotiation. *ACM Transactions on Autonomous and Adaptive Systems, 2,* 4 (November 2007), 14.
44. Plummer, D.C.; Smulders, C.; Fiering, L.; Natis, Y.V.; Mingay, S.; Driver, M.; Fenn, J.; McLellan, L.; and Wilson, D. Gartner's top predictions for IT organizations and users, 2008 and beyond. Stamford, CT: Gartner, 2008, www.gartner.com/it/page.jsp?id=593207.
45. Rinderle, S., and Benyoucef, M. Towards the automation of e-negotiation processes based on Web services. In A.H.H. Ngu, M. Kitsuregawa, E.J. Neuhold, J.-Y. Chung, and Q.Z. Sheng (eds.), *6th International Conference on Web Information Systems Engineering.* Berlin: Springer Verlag, 2005, pp. 443–453.
46. Ros, R., and Sierra, C. A negotiation meta strategy combining trade-off and concession moves. *Autonomous Agents and Multi-Agent Systems, 12,* 2 (March 2006), 163–181.
47. Rust, R.T., and Kannan, P.K. E-service: A new paradigm for business in the electronic environment. *Communications of the ACM, 46,* 6 (June 2003), 36–42.
48. Sandholm, T.W., and Lesser, V.R. Leveled commitment contracts and strategic breach. *Games and Economic Behavior, 35,* 1–2 (April 2001), 212–270.
49. Sim, K.M., and Choi, C.Y. Agents that react to changing market situations. *IEEE Transactions on Systems, Man and Cybernetics, Part B, 33,* 2 (April 2003), 188–201.
50. Sim, K.M., and Wang, S.Y. Flexible negotiation agent with relaxed decision rules. *IEEE Transactions on Systems, Man and Cybernetics, Part B, 34,* 3 (June 2004), 1602–1608.

51. Smith, J.M., and Price, G.R. The logic of animal conflict. *Nature, 246* (November 1973), 15–18.
52. Ströbel, M. Design of roles and protocols for electronic negotiations. *Electronic Commerce Research, 1,* 3 (July 2001), 335–353.
53. Su, S.Y.W.; Huang, C.; Hammer, J.; Huang, Y.; Li, H.; Wang, L.; Liu, Y.; Pluempitiwiriyawej, C.; Lee, M.; and Lam, H. An Internet-based negotiation server for e-commerce. *International Journal on Very Large Data Bases, 10,* 1 (August 2001), 72–90.
54. Tu, M. T.; Seebode, C.; Griffel, F.; and Lamersdorf, W. DynamiCS: An actor-based framework for negotiating mobile agents. *Electronic Commerce Research, 1,* 1–2 (February 2001), 101–117.
55. Umar, A. IT Infrastructure to enable next generation enterprises. *Information Systems Frontiers, 7,* 3 (July 2005), 217–256.
56. Vargo, S.L., and Lusch, R.F. Evolving to a new dominant logic for marketing. *Journal of Marketing, 68,* 1 (January 2004), 1–17.
57. Wainer, J.; Ferreira, P.R.; and Constantino, E.R. Scheduling meetings through multi-agent negotiations. *Decision Support Systems, 44,* 1 (November 2007), 285–297.
58. Wall, Q. Rethinking SOA governance. Oracle Inc., 2008, http://quintonwall.com/wp-content/uploads/2008/08/rethinking-soa-governance.pdf.
59. Zeng, D., and Sycara, K. Bayesian learning in negotiation. *International Journal Human-Computer Studies, 48,* 1 (January 1998), 125–141.
60. Zhu, L.; Leach, P.; Jaganathan, K.; and Ingersoll, W. The simple and protected generic security service application program interface (GSS-API) negotiation mechanism (RFC 4178). Network Working Group, October 2005, http://tools.ietf.org/html/rfc4178.

MANUEL RESINAS (resinas@us.es) is a lecturer on software engineering at the University of Seville, Spain, where he received his Ph.D. in 2008. His research interests include automated negotiation and automated analysis of service agreements and their relationship with business processes.

PABLO FERNÁNDEZ (pablofm@us.es) is a lecturer on software engineering at the University of Seville, where he is working toward his Ph.D. His research interests focus on automated trading.

RAFAEL CORCHUELO (corchu@us.es) is a reader in software engineering in the Department of Computer Languages and Systems of the University of Seville, where he received his Ph.D. He has been the leader of the university's Research Group on Distributed Systems since 1997. His research interests focus on the integration of Web data islands; previously, he worked on multiparty interaction and fairness issues.