SOCA
DOI 10.1007/s11761-008-0033-4

ORIGINAL RESEARCH PAPER

Agent-based communities of web services: an argumentation-driven approach

Jamal Bentahar · Zakaria Maamar · Wei Wan · Djamal Benslimane · Philippe Thiran · Sattanathan Subramanian

Received: 7 August 2007 / Revised: 17 August 2008 / Accepted: 22 September 2008

Abstract The objective of this paper is to discuss how to sustain the growth of Web services through the use of communities. A community aims at gathering Web services with the same functionality independently of their origins, locations, and functioning. To make Web services more responsive to the environment in which they run and to be more flexible when managing communities, Web services are associated with software agents enhanced with argumentation capacities. This type of agents persuade and negotiate with other peers for the sake of letting their respective Web services reach their goals in an efficient way. Associating Web services with this type of agents allows them to select good communities and allow the communities to host the good Web services and to select the best ones for composite scenarios. Furthermore, this provides satisfactory solutions for three open problems: starvation (Web services refuse all the possibilities of joining communities), competition-free (Web services accept joining any community without being selective), and unfairness (always the same Web services members of a community are selected out of many others to participate in composite scenarios). In addition, the paper presents a formal and computational persuasive and negotiation protocol to manage the attraction and retainment of Web services in the communities and their identification for composite services.

Keywords Community of Web services · Agents · Argumentation theory

J. Bentahar (✉)
Concordia University, Montreal, Canada
e-mail: bentahar@ciise.concordia.ca

Z. Maamar
Zayed University, Dubai, United Arab Emirates

W. Wan
Concordia University, Montreal, Canada

D. Benslimane
Claude Bernard Lyon 1 University, Lyon, France

P. Thiran
University of Namur, Namur, Belgium

S. Subramanian
INRIA Saclay-Île-de-France, Île-de-France, France

1 Introduction

1.1 Motivations

Recent years have seen an increasing interest in Web services. Indeed, they provide a suitable framework for developing largely-scale and loosely-coupled business processes. As the Web service technology is widely used for business applications and the number of developed Web services through the Internet continues to grow, the needs for discovering and engaging these Web services in more complex and complete business solutions are stressed out, leading to very rich settings for market competition. In [8], Bui and Gacher argue that although Web services are heterogeneous, the functionalities (e.g. `HotelBooking`) they offer are sufficiently well-defined and homogeneous enough to allow for market competition to happen. For example, the US National Digital Forecast Database XML Web Service (http://www.weather.gov/xml) and the US National Weather Service Forecast Office (http://www.srh.noaa.gov/mfl) propose each weather forecast-related Web services.

In this context, we proposed, along with other researchers, the notion of *communities of Web services* to gather Web services having the similar functionalities into special

structures [4,27,30,31,43]. Despite the exhaustive list of standards and specifications [23] (e.g. WSDL, SOAP, UDDI, WS-*) and projects on Web services [12,24,30,34], there is a lack of proper research that looks into the following community-related issues:

1. How to specify and represent communities of Web services in compliance with the existing standards and specifications?
2. How to initiate, set up, and dismantle a community of Web services? Who would be in charge of managing a community? And, how to specify and manage Web services in a community?
3. What kind of policies should guarantee the consistency of communities of Web services?
4. How to make a Web service select the community that suits it better? How to convince a Web service to stay longer in a community? And, how to eject a Web service from a community for misbehavior?
5. How to select a community member Web service out of many other members of the same community to take part of a given business scenario?

The aforementioned list of issues are open problems in the current research and development trend in the field of Web services. A promising way to address these problems and conduct research in this domain is making Web service more reactive and responsive to the environment in which they reside. Web services are still restricted to process users' or peers' requests without considering their internal execution states, or even questioning if they would be rewarded (e.g. will a Web service be privileged over other peers during selection?) for processing these requests [25]. In addition, some still consider Web services as passive components that react upon request only, which reduces and probably prevents their participation opportunities in complex business scenarios [7].

To make Web services more reactive to their environments and to allow them to interact more flexibly for the growing needs of business applications, we adopt the idea of empowering them with extra capabilities through the use of *software agents*. Briefly, software agents are autonomous entities, able to take initiatives in order to satisfy some goals [20]. In addition, agents are associated with metaphors and techniques that make them appropriate for designing, implementing, and verifying complex, distributed systems such as electronic trading and distributed business systems. Different formal logics can be used to specify and implement agents like epistemic, doxastic, and deontic [33,44]. The combination of Web services and software agents seems appealing and has already been studied in different research projects. In [22], Li et al. propose an agent framework to model and develop dynamic service-oriented operations. In [11], Dale et al. develop an evening organizer by combining Web services and agents. In [1] Baldoni et al. propose a logic programming-based approach to face the problem of automatic selection and composition of Web services. In this approach, Web services are viewed as software agents, communicating by predefined and sharable interaction protocols. A reasoning mechanism for performing the tasks of selection and composition of Web services in a way that is personalized w.r.t. the user request is provided. Last but not least, Maximilien and Singh present a multi-agent model for Web services and define catalogs of architectural styles for service-oriented architectures [28]. In this model, Web services are characterized as autonomous, policy-driven agents. By modeling Web services as agents, the authors augment the interaction sessions of Web services as interactions between and among service provider agents and service consumer agents, which captures a richer set of dynamic and autonomic interaction styles.

1.2 Contributions

The way we model and specify communities of Web services in this paper goes beyond the simple exercise of combining Web services and software agents. We aim at letting Web services initiate and engage in flexible interaction sessions based on their knowledge, interests, requirements, and needs. These sessions could be related to the community to join, the composition scenario to take part in, the peers to team up with, just to cite a few. This would shed the light on a new generation of Web services, which we label as *argumentative Web services*. Argumentative Web services will have the capability to argue, persuade, and negotiate with peers using a dialectical process when they want to affirm or disavow conclusions to convey to peers. For example, an argumentative Web service can persuade another Web service to join its community by exposing the advantages of being a member of such a community. Moreover, beyond a simple acceptance or rejection decision, an argumentative Web service can engage in more sophisticated interactions to negotiate the joining contract's clauses before making such a decision. Argumentative Web services are then services with more capabilities to reason about the commitments they could have in a community or before joining such a community.

This highlights an important contribution of the paper, which is providing sufficient solutions for problems that classical Web services cannot resolve. These problems are: (1) *starvation*, which means that without being argumentative, Web services can refuse all the invitations of joining communities if the offers are less than the expectations; (2) *competition-free*, which means that non argumentative Web services can accept joining any community without being selective; and (3) *unfairness*, which means that always the same Web services members of a community are selected

out of many others to participate in composite scenarios. The third problem can happen with non-argumentative Web services because when a decision should be taken to select a Web service for a composition after a call for proposals, Web services that are members of a community always bid with the same proposal, so the winner is always the same. When Web services are argumentative, negotiating joining offers and participation possibilities are possible. Furthermore, since Web services within a community can interact and coordinate, replacing each other if some execution problems arise is possible, which increases the system reliability. These benefits are implemented through an argumentation-based protocol for persuasion and negotiation that Web services can use in their interactions. We notice here that this protocol does not have to be integrated into existing Web services standards for the following reason. The protocol will empower the functioning of the agents that act on behalf of Web services. What is needed here is just linking software agents to Web services without affecting the way these Web services are already designed and deployed.

A community of argumentative Web services is not simply a society of software agents that come together in order to collaborate and achieve some common goals [35]. Although Web services in a community can, to a certain extent, collaborate in some situations, they generally compete to participate in composition scenarios since they all offer the same functionality but in a different set-up. Current approaches to design and develop Web services cannot handle this type of competition and argumentative agents are one more time appropriate for supporting Web services' operations. In such a competition setting, Web services should argue and provide evidences supporting the fact that they are better than others.

1.3 Paper's organization

The rest of this paper is structured as follows. In Sect. 2, we present the architecture of Web services communities, discuss the underlying management operations, and show through concrete scenarios the advantages of using argumentative Web services for these management operations. In Sect. 3, we introduce the argumentative agents we use to specify Web services and manage their respective communities. In Sect. 4, we present the argumentation-based framework for these communities and propose a persuasive negotiation protocol that Web services use to manage communities. In Sect. 5, we discuss the formal properties of the protocol along with its computational complexity. In Sect. 6, we present the system implementation along with some experimental results. Finally we conclude in Sect. 7. In the rest of this paper, argumentative Web service and argumentative software agent are used interchangeably.

2 Communities of Web services

2.1 Definitions and architecture

In Longman Dictionary, community is "*a group of people living together and/or united by shared interests, religion, nationality, etc.*" In the field of Web services, Benatallah et al. [4] define community as a collection of Web services with a common functionality independently of who provide them and how they function. Medjahed and Bouguettaya [31] consider community as a means to organize Web services that share the same domain of interest. Our definition goes beyond Web services gathering and considers community as a means to provide the description of a desired functionality (e.g. `FlightBooking`) without explicitly referring to any concrete Web service (e.g. `EKFlightBooking`) that will implement this functionality at run-time [26]. The definitions that Benatallah et al. and Medjahed and Bouguettaya suggest do not emphasize the dynamic nature of communities. A community should be able for instance to attract and retain the best Web services, to negotiate with the Web services the clauses of the membership contract, and to expel the Web services that do not fulfill their performance commitments. Examples of membership contract's clauses include initial rewards a community grants a Web service, participation rate by time unit a community guarantees to a Web service, leaving conditions and penalties, benefits a Web service is subject to after each participation in a composite scenario, etc. Current Web services' standards do not support these examples of operations, which shows the rationale of having argumentative agents acting on behalf of Web services and the communities they populate.

The development of communities of Web services offers some direct benefits to system designers in terms of accelerating the discovery process of Web services and enhancing Web services reliability at run-time [6].

1. The discovery process is now shifted to a higher level by targeting communities and not Web services. Current discovery practices are tied up to Web services and require screening several registries like UDDI or ebXML upon which Web services' descriptions are posted. These practices have proven to be tedious and require special techniques in terms of semantics and security [2,9,37]. In a composition scenario, the new discovery process would start by identifying appropriate communities according to users' needs and then, selecting from the identified communities the respective Web services to implement the required functionalities at run-time. This two-step process is almost similar to what Jureta et al. discuss in [21]. They suggest organizing Web services into *service centers*, which are specialized in delivering complete business-solutions based on Web services

composition. Contrarily to communities that are component Web services-centric, service centers are composite Web services-centric.

2. The reliability of Web services is enhanced and even sustained at run-time by allowing peers in a community to substitute each other without initiating the discovery process of other similar Web services from scratch, i.e. looking for communities and then for Web services [43]. If a Web service fails, the pool of replacement Web services, which have the same functionality like the failed one, is limited to the community hosting these Web services.

Figure 1 represents the architecture we developed to manage communities of Web services [27]. The components of this architecture include providers of Web services, UDDI registries (or any type of registry like ebXML), providers of Web services, and communities. A community is dynamic by nature. It is established and dismantled according to specific scenarios and protocols discussed in Sect. 2.2. UDDI registries receive advertisements of Web services from providers. Two communities of Web services are shown in Fig. 1. They could offer for example `FlightBooking` and `CarBooking` functionalities, respectively. A master component always leads a community. The master component could itself be implemented as a Web service for compatibility purposes with the rest of Web services that populate the community. These Web services are now denoted as slaves and have in common the functionality of the community to which they belong. Within the same community, slave Web services compete to participate in composition scenarios since they all implement the same functionality but in a different setting (e.g. different execution price, different response time).

One of the responsibilities of the master Web service is to attract Web services to sign up in its community using multiple types of rewards like better exposure to users during the discovery process, high probability of participation in composite scenarios, and benefits after each participation (Sect. 2.2). As a result, the master Web service screens the UDDI registries on a regular basis, so that it stays informed about the latest Web services' advertisements posted on these UDDI registries. Furthermore, a new Web service can contact the master Web service of a given community to express its interest in being part of this community based on the similarity that exists between its functionality and this community's functionality. An extra responsibility of the master Web service in a community is to nominate the slave Web service that will participate in a composite Web service. To this end and as suggested in [26], the master Web service runs the contract-net protocol [42] by sending all slave Web services a call for bids. This call for bids always comes along with the non-functional (i.e. QoS) criteria that the user sets for selecting Web services like response time and execution cost [32]. Prior to getting back to the master Web service, the slave Web services assess their status [25] and check their capacities of meeting these criteria. Only the slave Web services that are interested in bidding reply back to the master Web service. This latter screens all the bids before choosing the best one, e.g. a slave Web service's execution cost and reliability meeting the user's requirements. The winning slave Web service is notified and then it gets ready for execution when requested. The rest of the slave Web services that expressed interest but were not selected, are notified as well. However, if several slave Web services are selected because they offered the same bid, more sophisticated mechanisms are needed to choose one of them. Various solutions can be adapted for instance random selection, voting [39], or negotiation [45].

In a Web services community, the designation of a master Web service occurs in two different ways. The first way, which we choose in our work, is to have a dedicated Web service play the master role during the time being of a community. As a result, the master Web service never participates in compositions. The second way is to identify a slave Web service from the list of slave Web services that already populate a community. This identification could happen on a voluntary basis or after running election among the slave Web services. To keep the paper self-contained, additional details on master Web service designation are not provided.

2.2 Management

The management of a community of Web services revolves around the following questions: how to develop (dismantle) a new (existing) community, how to attract new Web services to be enrolled in an existing community, and how to retain existing Web services in a community? All these management operations are regulated through conversation sessions that argumentative Web services initiate.

2.2.1 Community of Web services development

A community is initially deployed to gather Web services with a similar functionality. This deployment is designer-driven and occurs in two steps. The first step is to define the functionality (e.g. `FlightBooking`) of the community by binding to a specific ontology [19]. This binding is crucial since providers of Web services use different terminologies to describe the functionality of their respective Web services. For example, `FlightBoo-king`, `FlightReservation`, and `AirTicketBooking` are all about the same functionality. The description of a Web service's functionality needs to be mapped onto the description of the functionality of the community using a specific ontology (i.e. ontology consists of concepts, axioms, relations, and instances).

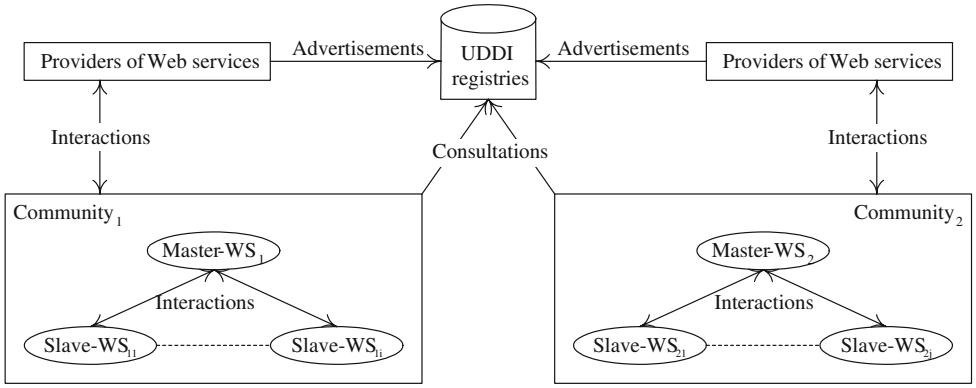

Fig. 1 Architecture of an environment of several independent communities of Web services

The second step in establishing a community is to deploy the master Web service of the community so that it takes over the multiple responsibilities we listed in Sect. 2.1. Some of these responsibilities include inviting Web services to sign up in its community and checking the credentials of Web services before they are admitted in its community. Credentials could be related to QoS (latency, execution time, privacy and security mechanisms, reliability, integrity, etc.), interaction protocols, interoperability, etc. Credential checking is critical to the reputation of a community as this boosts the security level in a community and enhances the trustworthiness level of a master Web service towards its slave Web services [17].

Dismantling a community of Web services is designer-driven as well that happens upon request from the master Web service. If this latter notices that the number of Web services in the community is less than a certain threshold and the number of participation requests in composite Web services that arrive from users over a certain period of time is less than another threshold, then the community will be dismantled. Both thresholds are set by the designer. A slave Web service that is ejected from a community is invited to join other communities subject to assessing the similarities between functionalities.

2.2.2 Web services attraction and retention

Attracting new Web services to a community and retaining the existing Web services in a community fall one more time under the responsibilities of the master Web service. We pointed out how a community of Web services could disappear if the number of residing Web services drops below a certain threshold. On the one hand, attracting Web services drives the master Web service to regularly consult the different UDDI registries looking for new Web services. These latter could have recently been posted on an UDDI registry or have seen their description changed. Changes in a Web service's description raise challenges since a Web service may no longer be appropriate for a community. As a result, this Web service is invited to leave the community. When a candidate Web service is identified in an UDDI registry according to its functionality, the master Web service interacts with this candidate. The purpose is to persuade the candidate Web service to register with its community. An argument that is used during this interaction is the high rate of participation of the existing Web services in composition scenarios, which is a good indicator of the visibility of a community to the external environment. Other arguments include short response-time in handling users' requests, and efficiency of the security mechanisms against malicious Web services.

On the other hand, retaining Web services in a community for a long period of time is a good indicator of the following elements:

- Although the Web services in a community are in competition, they expose a cooperative attitude. For instance, Web services are not subject to attacks from peers in the community. This backs the security argument that the master Web service uses to attract new Web services.
- A Web service is to a certain extent satisfied with its participation rate in composite Web services. This satisfaction rate is set by the provider of the Web service. In addition, this is inline with the participation-rate argument that the master Web service uses to attract new Web services.
- Web services are aware of peers in the community that could replace them in case of failure with less impact on the ongoing composite Web services in which they participate.

Web services attraction and retention shed the light on a third scenario. It is about Web services that are asked to leave a community. A master Web service could issue such a request upon assessment of the following criteria:

- The Web service has a new functionality, which does not perfectly match the functionality of the community.
- The Web service is unreliable. In different occasions, the Web service failed to participate in composite Web services due to recurrent operation problems.

– The credentials of the Web service were "beefed up" to enhance its participation opportunities in compositions. Ouzzani and Bouguettaya report that a Web service may not always fulfill its advertised QoS parameters due to various fluctuations related for example to the network status or resource availability [36]. Therefore, some differences between advertised and delivered QoS values occur. However, large differences indicate that the Web service is suffering a performance degradation and might not be able to sustain its advertised QoS.

2.3 Motivations behind argumentative Web services

Having discussed the management operations of a community of Web services, we provide hereafter some concrete scenarios that demonstrate the rationale of having argumentative Web services. In the architecture of Fig. 1, each master/slave Web service is associated with an argumentative agent that acts on its behalf. This should make Web services respond in a proper way to the various situations they could encounter at run-time. This association does not mean that agents from different service providers will reside and run on the same agent platform, but they can be deployed locally. Therefore, they can interact with others through Web services based message delivery and they can be easily re-configured without bothering distributed service advertisements.

Web services can initiate conversations with their peers before they decide to join/remain in/leave a community or participate in a composition scenario. These conversations could be based on previous ones and thus, can be used to reasons over them during persuasion and negotiation. Traditional Web services lack such reasoning capacities. As a result, their actions are very limited in reaction to the events happening in their environment. To illustrate the aforementioned discussion, let us consider the following two scenarios:

1. In the first scenario, the master Web service of a community identifies an appropriate Web service (in terms of functional and non-functional properties) in an UDDI and would like to invite it to its community. Let us assume that the following arguments are used to make the invitation attractive: (i) initial "monetary" reward the master suggests to the Web service if it accepts, and (ii) participation-rate range by time unit that the master promises to the Web service. To illustrate these arguments, figures are used such as $\texttt{InitialReward} = 100$ and $\texttt{ParticipationRate} \in [10, 20]$. In case of a non-argumentative Web service that is implemented as a set of hard-coded rules, it will either accept or refuse the membership offer. A rule of type $\texttt{InitialReward} \leq 140 \wedge \texttt{ParticipationRate} \notin [18, 25] \Rightarrow \mathit{Refuse\ Invitation}$, (where $\wedge$ and $\Rightarrow$ stand for "and" and "then", respectively) will make the Web service refuse the offer. This simple example sheds the light on a "starvation" problem where a Web service will automatically turn down all the invitations because of the gap between the offers and its expectations. To handle this scenario, the designer of the Web service has to review the rules. Moreover, the Web service can continue refusing offers even though they are close to the limits imposed by the designer as the following rule illustrates: $\texttt{Initial Reward} \leq 101 \wedge \texttt{ParticipationRate} \notin [10, 20] \Rightarrow \mathit{RefuseInvitation}$. If the thresholds are low enough and interval ranges are large enough to avoid this "starvation" problem (i.e. $\texttt{InitialReward} \leq 70 \wedge \texttt{ParticipationRate} \notin [4, 30]$), then the Web service will accept the first offer even if it deserves a much better offer considering its outstanding QoS. Accepting any offer, without being selective, will create a "competition-free" environment between Web services. Plus, there will be no distinction between "good" and "poor" Web services. Finding a trade-off solution in order to avoid the "starvation" and "non-competitiveness" problems is not straightforward when rules that dictate the operations of Web services are rigid and hard-coded.
   If Web services were argumentative, they could address the "starvation" problem. They could consider previous interactions with different masters and review (e.g. lowering) their expectations according to the available offers. Moreover, they could manage the "non-competitiveness" problem by reasoning over different offers and negotiating them with the master in order to maximize their benefits.
   In the aforementioned scenario, the number of arguments (i.e. `InitialReward` and `ParticipationRate`) was limited to two, only. In real life, multiple arguments could be used, such as contract duration, departure conditions, QoS to maintain (mean response time (latency), execution time, transaction time, throughput, pricing policy, privacy and security mechanisms, reliability, integrity, etc.), reputation of the community, availability, etc. In such a scenario, the use of hard-coded rules will accentuate the "starvation" and "non-competitiveness" problems. Contrarily, argumentative Web services could smoothly engage in negotiation sessions for the sake of achieving better agreements. Therefore, rewards could be subject to the QoS that Web services provide. For example, if a Web service's response time was short, and its reliability was high, then the reward should be high.
2. In the second scenario, the focus is on composition. We mentioned that the master Web service uses the contract-net protocol to select the Web service that will participate in a composite Web service (Sect. 2.1). On the one hand, in case of non-argumentative Web services, these ones will always provide the same bids based on

their hard-coded rules every time a similar call for bids is received. Therefore, the same Web service, or the same set of Web services, will be selected. This will lead to another problem namely "unfairness" since other Web services will automatically be discarded. In addition, if many Web services submit the same bid (e.g. same QoSs and same fares), the non-argumentative master will select one of them either randomly or by voting. This mode of selection has several drawbacks. Although voting insures fairness *versus* random selection, implementing it with competitor and "selfish" Web services is proved to be inefficient [13]. Furthermore, using random and voting selection without giving Web services room to interact with each other will deprive them from collaboration. As a result, they cannot replace each other if some problems arise in run-time.

On the other hand, i.e. with argumentative Web services, the "unfairness" problem can be addressed by changing the biding strategies if after a while Web services finds out that their bids continue to be unselected compared to what other peers provide. The selection problem can then be fixed by allowing argumentative Web services to negotiate their participation with each other. An argument to use in this negotiation is resource and benefit sharing. As an outcome example of the negotiation, one of them is selected subject to selecting the other one next time.

Empowering Web services with argumentation capabilities comes with its own set of challenges. For instance, the high computational complexity of argumentation-based reasoning and decision making procedures (as reported in [38]) might affect the performance of Web services. To address this challenge, we show in Sect. 5 that the use of a restricted logical language for instance *Horn logic*, and simple logical rules should be enough to implement the operations related to community management such as attraction, retention, and selection.

3 Argumentative agents

Prior to explaining the use of argumentation to manage Web services communities, an overview of this theory is deemed appropriate.

Argumentation is defined as a dialectical process for the interaction of different arguments for and against some conclusions [18,41]. Argumentation can help multiple agents make decisions, interact rationally, convince and negotiate with peers by relying on reasons that lead into conclusions [16]. Furthermore, argumentation can assist agents to internally perform their reasoning when they control and require access to resources and services.

At the architectural level, agents using argumentation (called argumentative agents) are BDI agents augmented with additional capabilities that make them more autonomous. Indeed, such agents can argue about their beliefs and goals, and build and evaluate arguments before making decisions. To be able to negotiate, BDI (non-argumentative) agents use static and predefined protocols and game theory-based techniques. Predefined protocols are not flexible and they can be used only for basic interactions where the outcomes are deterministic, and using game theory assumes, among other things, that agents have unbounded computational resources, have complete knowledge of the outcome space, and are optimizers of utility in the sense of rational-choice theory. From a computational perspective, these are unrealistic assumptions for automatic negotiation. Agents can also use an heuristic approach for negotiation. Such an approach will try to maximize the utility of each agent by making iterative concessions until an agreement is achieved. However, such a solution can be used only for simple, one issue negotiation. For multi-issue negotiations involving several parameters, an heuristic approach cannot guarantee achieving an agreement satisfying both agents. This is because multi-issue negotiation needs more sophisticated reasoning capabilities to reason about different combinations. Another limit of heuristic approaches is the fact that agents' strategies are rigid and cannot be changed when new information becomes available. In fact, negotiation takes place when there is a conflict and agents are willing to collaborate in order to achieve an agreement. Argumentation seems a natural solution since by providing arguments that justify the offers, there is a high chance of achieving an agreement resolving the conflict. In Sect. 5.1, we prove that if such an agreement is theoretically possible considering the union of the agents' knowledge bases, then by using a suitable argumentation protocol, this agreement will always be achieved. This solution can be used for simple as well as complex negotiation. For the purpose of managing communities of Web services, multi-issue negotiations are most likely to happen particularly when negotiating the joining contract involving several terms such as the initial reward, participation rate in composite scenarios, benefits gained after each participation, contract duration, leaving conditions and penalties, etc. For this reason, argumentation seems to be the most suitable solution for this type of negotiation.

Several argumentation theories and frameworks have been proposed in the literature like [6,10,41]. An argumentation system essentially includes a logical language $\mathcal{L}$, a definition of the *argument concept*, a definition of the *attack relation* between arguments, and a definition of the *acceptability of arguments*. The use of a logical language enables argumentative agents to use a logic-based reasoning in order to effectively reason about arguments in terms of inferring and justifying conclusions, and attacking and defending

arguments. The attack relation is based on a contrary relation, which is not necessarily symmetric. For example, a contrary of the fact that "the response time of the Web service falls into the interval [2 ms, 10 ms]" is "the response time of this Web service is in [8 ms, 14 ms]." A contrary of a formula p is denoted by $\bar{p}$. The logical negation $\neg$ is an example of the contrary relation. Hereafter, we define the concepts that will be used in the framework for managing communities of Web services ($\vdash$ stands for classical inference and $\rightarrow$ for material implication):

Definition 1 (*Argument*) Let Γ be a knowledge base with no deductive closure. An argument is a pair (H, h) where h is a formula of $\mathcal{L}$ and H a subset of Γ such that: (i) H is consistent, (ii) $H \vdash h$, and (iii) H is minimal, so that no subset of H satisfying both (i) and (ii) exists. H is called the support of the argument and h its conclusion.

Example 1 Let $\Gamma = \{a, b, t, a \wedge b \rightarrow c, c \wedge d \rightarrow m, \neg m, r, \neg a, r \rightarrow d, t \rightarrow \neg c\}$.
$(\{a, b, a \wedge b \rightarrow c, r, r \rightarrow d, c \wedge d \rightarrow m\}, m)$ is an argument supporting m.

Definition 2 (*Attack relation*) Let $\mathcal{AT}$ be a binary relation between arguments, and (H, h) and (H', h') be two arguments. (H', h') $\mathcal{AT}$ (H, h) iff $H' \vdash \bar{h}$ or $\exists x : H \vdash x$ and $h' \equiv \bar{x}$. In other words, an argument is attacked if and only if there exists an argument that contradicts its conclusion or an element of its support.

Example 2 Considering the same Γ as in Example 1, the argument: $(\{t, t \rightarrow \neg c\}, \neg c)$ attacks the argument $(\{a, b, a \wedge b \rightarrow c, r, r \rightarrow d, c \wedge d \rightarrow m\}, m)$ since $(\{a, b, a \wedge b \rightarrow c, r, r \rightarrow d, c \wedge d \rightarrow m\} \vdash c$. Also the argument $(\{\neg m\}, \neg m)$ attcks the argument $(\{a, b, a \wedge b \rightarrow c, r, r \rightarrow d, c \wedge d \rightarrow m\}, m)$.

An argument is acceptable if it belongs to an acceptable set of arguments. A set of arguments is acceptable if it is conflict free (there is no two arguments in the set such that one attacks the other) and defends all its elements against all the attackers (see [15] for more details about the different acceptability semantics).

4 Persuasive negotiation protocol for communities of Web services

4.1 Formal foundation

The characteristics of argumentation-based agents discussed in Sect. 3 make them suitable for modeling dynamic and proactive Web services. The primary added value is to let Web services interact and argue with each other before joining and settling down afterwards in a community. They will be able to reason about and compare different joining offers in order to maximize their benefits. For example, Web services can select the best community in terms of reputation and get the best reward and the best participation rate. In addition, they can maximize their performance within the community by negotiating with peers the participation conditions in composite scenarios. They can share participation benefits and collaborate by sharing resources and replacing each other if some problems arise at run-time [3]. In fact, as discussed in Sect. 2.3, argumentative Web services provide solutions to the "starvation," "non-competitiveness," "unfairness," and "selection" problems, which cannot be resolved using traditional Web services equipped with rigid, hard-coded rules.

Our argumentative agents act as representatives to Web services, reason on their behalf, and identify situations that maximize their profits (e.g. participation-rate increase) and minimize their expenses (e.g. resource consumption decrease). Metadata describing Web services in terms of contents and features are represented within the state of the agents. Web services within communities are connected through a communication network so that they can interact and share resources to reach some joint goals.

In this section, we present a persuasive negotiation protocol that allows argumentative Web services to negotiate the contract of joining a community and their participation in composite scenarios. We present first the general protocol form and then illustrate its instantiation for a scenario about joining a community. To reason about Web services and communities, argumentative agents are equipped with knowledge, beliefs, and argumentation capabilities. The agent of a Web service Ag_{WS} knows all details on its Web service in terms of functionality, QoS, (mean response time, execution time, transaction time, throughput, reliability, integrity, etc.), and any other relevant details. The knowledge base of Ag_{WS} is denoted by $KB(Ag_{WS})$. An argumentative Web service can also have beliefs towards other Web services whether in the same community or in other communities. Descriptions of these Web services, their functionalities, QoS, and trust are examples of these beliefs. As explained in the previous section, the agent's argumentation system is built upon the agent's beliefs and knowledge.

In [38], Parsons et al. prove that argumentation reasoning procedures based on languages expressed in the first order logic or even propositional logic are computationally intractable. For communities of Web services, a restricted language such as *Horn logic* is enough to represent agents' beliefs and to develop their reasoning capacities. Horn logic is expressed in terms of *propositional Horn clauses*. Such a clause is a disjunction of literals with at most one positive literal $\neg p_1 \vee \neg p_2 \vee \cdots \vee \neg p_n \vee c$ (also written as implication $p_1 \wedge p_2 \wedge \cdots \wedge p_n \rightarrow c$). *A propositional Horn formula* is a conjunction of propositional Horn clauses. These clauses could be restricted to be *definite* where each clause has exactly one

positive literal. *A propositional definite Horn formula* is a conjunction of propositional definite Horn clauses. This restriction is of a particular interest in modeling argumentative reasoning, since formulas of type $p_1 \wedge p_2 \wedge \cdots \wedge p_n \rightarrow c$ are adequate to describe interrelationships between premises and conclusions. This could be used to support positive literals. For example, the master Web service can have an argument supporting the conclusion that the community it is leading has a good reputation, which can be represented as follows: NWS = High $\wedge$ NUR = High $\wedge$ NSC = High $\wedge$ NLW = Low $\rightarrow$ Reputation = Good where NWS stands for Number of current Web Services in the community, NUR for Number of Users' Requests the community receives by time unit, NSC for Number of Successful Compositions by time unit to which the community's Web services participate, and NLW for Number of Leaving Web services. Also, a slave Web service can have an argument supporting the fact that it deserves a better offer, which can be represented as follows: $R = V_1 \wedge R\_C_x = V_2 \wedge V_2 \geq V_1 \wedge \text{QoS} = \text{Good} \rightarrow \text{DR} \geq V_1$ where R stands for the Reward offered by the community the Web service is negotiating with, R_C_x for the Reward another Community (called C_x) already offered to this Web service, and DR for Deserved Reward. In other words, when negotiating with a given community, if an argumentative Web service has a good QoS and already received a competitive offer from another community, then it has an argument to ask for a better offer. In addition, for Web services, there is no need to suppose that the knowledge bases are inconsistent. The reason is that the size of these knowledge bases are generally small enough, so that checking the consistency when a new belief is added becomes doable in a linear time.

4.2 General specification

To be able to persuade an argumentative Web service to join a community or to remain in a community, and to negotiate the participation in a given composite scenario along with the outcome of the contract-net protocol, the master and slave argumentative Web services use persuasive negotiation techniques based upon their argumentation abilities. Hereafter, we specify a Horn logic-based protocol to use for these persuasion and negotiation activities. This protocol is specified as a combination of a set of initiative/reactive *dialogue games*. Dialogue games can be thought of as interaction games in which each Web service plays a move in turn by performing utterances according to a pre-defined set of rules [29]. Dialogue games have the advantage of being more flexible than classical protocols such as FIPA-ACL protocols.[1] Indeed, a dialogue game can be specified as a combination of small conversation policies that Web services can combine by reasoning over them using a set of logical rules [5]. From a logical point of view, game moves are considered as communicative acts that argumentative Web services perform. Formally, we define a protocol for argumentative Web services as follows:

Definition 3 (*Protocol*) A protocol Pr for argumentative Web services is a tuple $\langle \mathcal{C}, \mathcal{D} \rangle$ where $\mathcal{C}$ is a finite set of allowed communicative acts and $\mathcal{D}$ is a set of dialogue games.

The allowed communicative acts in our persuasive negotiation protocol are: *Open*, *Accept*, *Refuse*, *Make-Offer*, *Challenge*, *Justify*, and *Attack*. *Open* is a special communicative act used to open the protocol. The type of a communicative act refers to its name, for example *Accept* is a communicative act of type *Accept*. We define a dialogue game in our protocol as follows:

Definition 4 (*Dialogue game*) Let $CA_i(Ag_{WS_1}, Ag_{WS_2}, p)$ be a communicative act of type i performed by an argumentative Web service Ag_{WS_1} and sent to another argumentative Web service Ag_{WS_2} about a content p, and $CA_{i_j}(Ag_{WS_2}, Ag_{WS_1}, p')$ be the communicative act of type j with content p' that depends on the communicative act of type i. A dialogue game Dg is a conjunction of rules, where each rule identifies one possible communicative act that a Web service can use as a reply when receiving a communicative act from another Web service if a given condition C_{i_j} is satisfied. This conjunction is specified as follows:

$$\bigwedge_{0<j\leq n} \big(CA_i(Ag_{WS_1}, Ag_{WS_2}, p) \wedge C_{i_j} \Rightarrow CA_{i_j}(Ag_{WS_2}, Ag_{WS_1}, p')\big)$$

where $\Rightarrow$ is the implication symbol for dialogue game rules, and n is the number of allowed communicative acts that Ag_{WS_2} can perform after receiving a communicative act from Ag_{WS_1}.

We use the symbol $\Rightarrow$ instead of $\rightarrow$ to distinguish the implication in the dialogue game rules from the one used in Horn formulas. In this definition, content p could be a Horn formula or an argument expressed in Horn clauses. C_{i_j} is expressed in terms of the possibility of generating an argument from the argumentation system.

The formula $p \triangleleft Arg\_Sys(Ag_{WS})$ expressed in Horn language $\mathcal{L}$ denotes the fact that a Horn propositional formula p can be generated from the Ag_{WS}'s argumentation system denoted by $Arg\_Sys(Ag_{WS})$. The formula $\neg(p \triangleleft Arg\_Sys(Ag_{WS}))$ indicates the fact that p cannot be generated from Ag_{WS}'s argumentation system. For example, if the master Web service $Ag_{MWS}\_C_x$ of a community C_x has an argument for the fact that the reputation of its community is good, this will be represented by:

$$Reputation\_C_x = Good \triangleleft Arg\_Sys(Ag_{MWS}\_C_x)$$

A Horn propositional formula p can be generated from an argumentative Web service's argumentation system, if this

[1] http://www.fipa.org.

Web service can build an argument supporting p using its argumentation system. The following is an example of a dialogue game, in which a master Web service $Ag_{MWS1}\_C_x$ of a community C_x invites a Web service Ag_{WS2} to join the community.

Example 3

$$Open(Ag_{MWS1}\_C_x, Ag_{WS_2}, p)$$
$$\wedge \big(\neg(Reputation\_C_x = Poor \triangleleft Arg\_Sys(Ag_{WS_2}))$$
$$\wedge(Joining\_Commitment = false \triangleleft Arg\_Sys(Ag_{WS_2}))\big)$$
$$\Rightarrow Accept(Ag_{WS_2}, Ag_{MWS1}\_C_x, p)$$
where $p = Invitation\_for\_Joining\_C_x$

In this example, Ag_{WS_2} accepts the invitation because it does not have any argument supporting the fact that the reputation of the community is "poor" and it has an argument that it is not committed to join any other community ($Joining\_Commitment = false$). Accepting the invitation does not mean that Ag_{WS_2} commits to join the community, but only a negotiation of the joining contract can start.

4.3 Argumentative dialogue games for communities of Web services

In our persuasive negotiation protocol for argumentative Web services, we distinguish three types of dialogue games (Fig. 2): *Entry game*, *Chaining games*, and *Termination game*. The *Entry game* enables conversation opening and setting up. The *Chaining games* make it possible to continue the conversation by combining several dialogue games. The persuasive negotiation protocol includes four chaining dialogue games: *Offer game*, *Challenge game*, *Attack game* and *Justifica-tion game*. The conversation terminates when the exit conditions are satisfied (*Termination game*).

4.3.1 Entry game

General form

The *Entry game* allows Web services to initiate conversations. For example, if a master Web service decides to invite a new Web service registered in a given UDDI to be a member of its community, this master will trigger an *Entry game* with invitation to join the community as subject. If the new Web service accepts, then the master can suggest rewards to the Web service if its final decision is to join the community. If the Web service refuses the invitation, the protocol terminates. A Web service can turn down invitations if it is not interested in a community (e.g. low participation-rate of existing Web services, or decided to join another community). Within a same community, a Web service can invite other Web services to negotiate their participation in a composite Web service. This occurs, as mentioned in Sect. 2, when several agents provide the same "winning" bid following the

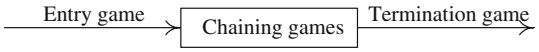

Fig. 2 Types of dialogue games

master Web service's call for bids. The master asks one of the winnings to invite others for negotiation. The negotiation starts upon receiving and accepting this invitation. We specify the *Entry game* as follows:

$$\big(Open(Ag_{WS_1}, Ag_{WS_2}, p) \wedge C_1$$
$$\Rightarrow Accept(Ag_{WS_2}, Ag_{WS_1}, p)\big)$$
$$\wedge\big(Open(Ag_{WS_1}, Ag_{WS_2}, p) \wedge C_2$$
$$\Rightarrow Refuse(Ag_{WS_2}, Ag_{WS_1}, p)\big)$$

where:

$$C_1 = (p \triangleleft Arg\_Sys(Ag_{WS_2})) \vee \neg(\neg p \triangleleft Arg\_Sys(Ag_{WS_2}))$$
$$C_2 = \neg p \triangleleft Arg\_Sys(Ag_{WS_2})$$

Proposition p is expressed in the logical language $\mathcal{L}$ using a shared ontology. This proposition indicates an invitation to start a conversation. If the invited Web service has an argument in favor of p or does not have any argument against p, it accepts the invitation. Otherwise, it refuses. For example, if a new Web service is not interested in joining a community due to previous unsuccessful experiences in this community, a refusal is sent to the master Web service. If a Web service believes that the community's configuration is efficient (i.e. good reputation and high participation rate), and no commitment is made to join any other community, then it will accept the invitation. The following example instantiates the *Entry game* with respect to this invitation scenario.

Example 4

$$\big(Open(Ag_{MWS1}\_C_x, Ag_{WS_2}, p)$$
$$\wedge\big((Reputation\_C_x = Good \triangleleft Arg\_Sys(Ag_{WS_2}))$$
$$\wedge(Participation\_Rate\_C_x = High \triangleleft Arg\_Sys(Ag_{WS_2}))$$
$$\wedge(Joining\_Commitment = false \triangleleft Arg\_Sys(Ag_{WS_2}))\big)$$
$$\Rightarrow Accept(Ag_{WS_2}, Ag_{MWS1}\_C_x, p)\big)$$
$$\wedge\big(Open(Ag_{MWS1}\_C_x, Ag_{WS_2}, p)$$
$$\wedge\big((Reputation\_C_x = Poor \triangleleft Arg\_Sys(Ag_{WS_2}))$$
$$\vee(Participation\_Rate\_C_x = Low \triangleleft Arg\_Sys(Ag_{WS_2}))$$
$$\vee(Joining\_Commitment = true \triangleleft Arg\_Sys(Ag_{WS_2}))\big)$$
$$\Rightarrow Refuse(Ag_{WS_2}, Ag_{MWS1}\_C_x, p)\big)$$
where $p = Invitation\_for\_Joining\_C_x$

4.3.2 Offer game

General form

Once the *Entry game* is accepted, the initiator argumentative Web service starts by making an offer. In the case of inviting a new Web service to join a community, the offer contains the initial rewards that the master offers to the new Web service and the advantages of being a member of this community.

In the case of persuading an existing Web service to remain in the community, the offer could include increases in the rewards. In the case of negotiating a participation in a composite Web service, the offer contains the rewards that the initiator will give to the other Web service after the composition, for example, a part of the rewards this initiator will obtain after its participation. Let p and q be two Horn formulas representing offers (contents of an *Offer* communicative act). The notation $p \simeq q$ indicates the fact that these two offers are for the same object. We specify the *Offer game* as follows:

$$
\begin{aligned}
&\big(\mathit{Make\text{-}Offer}(Ag_{WS_1}, Ag_{WS_2}, p) \wedge C_1 \Rightarrow \\
&\quad \mathit{Accept}(Ag_{WS_2}, Ag_{WS_1}, p)\big) \\
\wedge&\big(\mathit{Make\text{-}Offer}(Ag_{WS_1}, Ag_{WS_2}, p) \wedge C_2 \Rightarrow \\
&\quad \mathit{Challenge}(Ag_{WS_2}, Ag_{WS_1}, p')\big) \\
\wedge&\big(\mathit{Make\text{-}Offer}(Ag_{WS_1}, Ag_{WS_2}, p) \wedge C_3 \Rightarrow \\
&\quad \mathit{Attack}(Ag_{WS_2}, Ag_{WS_1}, (H, \neg p'))\big) \\
\wedge&\big(\mathit{Make\text{-}Offer}(Ag_{WS_1}, Ag_{WS_2}, p) \wedge C_4 \Rightarrow \\
&\quad \mathit{Make\text{-}Offer}(Ag_{WS_2}, Ag_{WS_1}, q)\big) \\
\wedge&\big(\mathit{Make\text{-}Offer}(Ag_{WS_1}, Ag_{WS_2}, p) \wedge C_5 \Rightarrow \\
&\quad \mathit{Refuse}(Ag_{WS_2}, Ag_{WS_1}, p)\big)
\end{aligned}
$$

where

$$
\begin{aligned}
&C_1 = p \lhd \mathit{Arg\_Sys}(Ag_{WS_2}) \\
&C_2 = \exists p' \subseteq p : \\
&\neg(p' \lhd \mathit{Arg\_Sys}(Ag_{WS_2})) \wedge \neg(\neg p' \lhd \mathit{Arg\_Sys}(Ag_{WS_2})) \\
&C_3 = \exists p' \subseteq p : \\
&(H \lhd \mathit{Arg\_Sys}(Ag_{WS_2})) \wedge (H, \neg p')\ \mathcal{AT}\ (p, p) \\
&C_4 = p \simeq q \wedge q \lhd \mathit{Arg\_Sys}(Ag_{WS_2}) \\
&C_5 = \neg(C_1 \vee C_2 \vee C_3 \vee C_4)
\end{aligned}
$$

By definition, $\mathit{Attack}(Ag_{WS_2}, Ag_{WS_1}, (H, \neg p'))$ means that Ag_{WS_2} asserts argument $(H, \neg p')$ to attack a part or the whole offer proposed by Ag_{WS_1}. The generation of a set of formulae H from Ag_{WS_2} is defined as follows:

$$
H \lhd \mathit{Arg\_Sys}(Ag_{WS_2}) \stackrel{\triangle}{=} \forall h_i \in H\ h_i \lhd \mathit{Arg\_Sys}(Ag_{WS_2})
$$

When an argumentative Web service receives an offer, it accepts it if it has a supporting argument for it, challenges a part of it if it has no argument for or against this part, attacks if it has an argument against the offer, and/or makes a counter offer if it can generate such a counter-offer from its knowledge base using its argumentation system. The content of an attack could also be a counter-offer. If none of these conditions is satisfied, the addressee refuses the offer. For example, an argumentative Web service can refuse an offer if it has already accepted a different offer made by another master Web service about the same subject. Generally, within a dialogue game, an argumentative Web service can only play one move. However, *Attack* and *Make-Offer* moves can be played together; attacking offers and then making counter-offers.

Example 5 Let us suppose that the master $Ag_{MWS1\_C_x}$ and the Web service Ag_{WS_2} would like to negotiate two terms in the joining contract: the Ag_{WS_2}'s availability increasing and participation rate in composite business scenarios the master can provide. By the availability we mean the availability of the Web service as perceived by the user. The master can promise to Ag_{WS_2} to increase its availability since the community provides a collaborative setting so it can share resources with other Web services and they can replace it in case of failure execution. Let $p(v_1, v_2)$ indicate the fact that the availability will be increased by a value v_1 and the participation rate by time unit will be of a value v_2. An instantiation of the the *Offer game* is as follows:

$$
\begin{aligned}
&\big(\mathit{Make\text{-}Offer}(Ag_{MWS1\_C_x}, Ag_{WS_2}, p(v_1, v_2)) \\
&\quad \wedge\ p(v_1, v_2) \lhd \mathit{Arg\_Sys}(Ag_{WS_2}) \\
&\quad \Rightarrow \mathit{Accept}(Ag_{WS_2}, Ag_{MWS1\_C_x}, p(v_1, v_2))\big) \\
\wedge&\big(\mathit{Make\text{-}Offer}(Ag_{MWS1\_C_x}, Ag_{WS_2}, p(v_1, v_2)) \\
&\quad \wedge\ \neg(p(v_1, v_2) \lhd \mathit{Arg\_Sys}(Ag_{WS_2})) \\
&\quad \wedge\ \neg(\neg p(v_1, v_2) \lhd \mathit{Arg\_Sys}(Ag_{WS_2})) \\
&\quad \Rightarrow \mathit{Challenge}(Ag_{WS_2}, Ag_{MWS1\_C_x}, p(v_1, v_2))\big) \\
\wedge&\big(\mathit{Make\text{-}Offer}(Ag_{MWS1\_C_x}, Ag_{WS_2}, p(v_1, v_2)) \\
&\quad \wedge\ (\mathit{Av\_WS_2} + v_1 < \mathit{Availability\text{-}C_x} \lhd \mathit{Arg\_Sys}(Ag_{WS_2})) \\
&\quad \wedge\ (v_2 < \mathit{Participation\text{-}C_x} \lhd \mathit{Arg\_Sys}(Ag_{WS_2})) \\
&\quad \Rightarrow \mathit{Make\text{-}Offer}(Ag_{WS_2}, Ag_{MWS1\_C_x}, p(\mathit{Availability\text{-}} \\
&\qquad C_x + \epsilon_1, v_2 + \epsilon_2))\big) \\
\wedge&\big(\mathit{Make\text{-}Offer}(Ag_{MWS1\_C_x}, Ag_{WS_2}, p(v_1, v_2)) \\
&\quad \wedge\ \neg p(v_1, v_2) \lhd \mathit{Arg\_Sys}(Ag_{WS_2}) \\
&\quad \Rightarrow \mathit{Refuse}(Ag_{WS_2}, Ag_{MWS1\_C_x}, p(v_1, v_2))\big)
\end{aligned}
$$

In this example, Ag_{WS_2} accepts the offered availability increasing and participation rate if it has an argument supporting them, which means that the offer fits the Web service's expectation. If no argument can support the offer or attack it, Ag_{WS_2} asks for justification. This means that the Web service does not have any expectation regarding the availability increasing and the participation rate. For example, the Web service does not know if the offered participation rate is more or less than the community's average. If the increasing availability and participation rate are not enough because they are less than the community's average, Ag_{WS_2} will make a counter-offer. In the example, ϵ_1 and ϵ_2 can be calculated so that the new offer fits the Ag_{WS_2}'s expectation. Otherwise, the offer will be simply refused. For simplicity reasons, the attack option is merged with the counter-offer in this example.

4.3.3 Challenge and Justification games

General form

The *Challenge game* is specified as follows:

$$
\begin{aligned}
&\mathit{Challenge}(Ag_{WS_1}, Ag_{WS_2}, p) \wedge C_1 \\
&\quad \Rightarrow\ \mathit{Justify}(Ag_{WS_2}, Ag_{WS_1}, (H, p))
\end{aligned}
$$

where

$$C_1 = H \triangleleft Arg\_Sys(Ag_{WS_2})$$

Condition C_1 should always be satisfied since a Web service must always be able to justify its propositions and assertions.

We specify the *Justification game* as follows:

$$\begin{aligned}
&\big(Justify(Ag_{WS_1}, Ag_{WS_2}, (H, p)) \wedge C_1 \\
&\quad \Rightarrow Accept(Ag_{WS_2}, Ag_{WS_1}, H)\big) \\
&\wedge\big(Justify(Ag_{WS_1}, Ag_{WS_2}, (H, p)) \wedge C_2 \\
&\quad \Rightarrow Challenge(Ag_{WS_2}, Ag_{WS_1}, H')\big) \\
&\wedge\big(Justify(Ag_{WS_1}, Ag_{WS_2}, (H, p)) \wedge C_3 \\
&\quad \Rightarrow Attack(Ag_{WS_2}, Ag_{WS_1}, (H', p'))\big) \\
&\wedge\big(Justify(Ag_{WS_1}, Ag_{WS_2}, (H, p)) \wedge C_4 \\
&\quad \Rightarrow Make\text{-}Offer(Ag_{WS_2}, Ag_{WS_1}, q))\big) \\
&\wedge\big(Justify(Ag_{WS_1}, Ag_{WS_2}, (H, p)) \wedge C_5 \\
&\quad \Rightarrow Refuse(Ag_{WS_2}, Ag_{WS_1}, p))\big)
\end{aligned}$$

where

$$\begin{aligned}
&C_1 = H \triangleleft Arg_Sys(Ag_{WS_2}) \\
&C_2 = \exists H' \subseteq H : \forall h_i \in H' \\
&\quad \neg(h_i \triangleleft Arg_Sys(Ag_{WS_2})) \wedge \neg(\neg h_i \triangleleft Arg_Sys(Ag_{WS_2})) \\
&C_3 = H' \triangleleft Arg_Sys(Ag_{WS_2})) \wedge (H', p')\ \mathcal{AT}\ (H, p) \\
&C_4 = p \simeq q \wedge q \triangleleft Arg_Sys(Ag_{WS_2}) \\
&C_5 = \neg(C_1 \vee C_2 \vee C_3 \vee C_4)
\end{aligned}$$

Challenging a set of formulae H means that challenging all the formulas in it:

$$\begin{aligned}
&Challenge(Ag_{WS_2}, Ag_{WS_1}, H) \triangleq \forall h_i \in H \\
&\quad Challenge(Ag_{WS_2}, Ag_{WS_1}, h_i)
\end{aligned}$$

These five conditions are similar to those associated with the *Offer game*. The only difference in the *Justifi-cation game* resides in *Accept* and *Attack* moves that are relative to the support of the offer and not to the offer itself.

Example 6 Let us continue the scenario we instantiated in Example 5 in which, the master $Ag_{MWS1\_}C_x$ and the Web service Ag_{WS_2} negotiate the Ag_{WS_2}'s availability increasing and participation rate in composite business scenarios. When Ag_{WS_2} challenges the offer, $Ag_{MWS1\_}C_x$ justifies it by the fact that it is greater than the community's average. Ag_{WS_2} attacks then the answer by providing an example of another community C_y providing the same functionalities but with a higher participation rate. This scenario is formulated as follows:

$$\begin{aligned}
&\big(Challenge(Ag_{WS_2}, Ag_{MWS_1_}C_x, p(v_1, v_2)) \\
&\quad \wedge (Av_WS_2 + v_1 > Availability\text{-}C_x \triangleleft Arg_Sys(Ag_{MWS_1_}C_x)) \\
&\quad \wedge (v_2 > Participation\text{-}C_x \triangleleft Arg_Sys(Ag_{MWS_1_}C_x)) \\
&\quad \Rightarrow Justify(Ag_{MWS_1_}C_x, Ag_{WS_2}, (H, p(v_1, v_2)))\big) \\
&\big(Justify(Ag_{MWS_1_}C_x, Ag_{WS_2}, (H, p(v_1, v_2))) \\
&\quad \wedge (Participation\text{-}C_y > v_2 \triangleleft Arg_Sys(Ag_{WS_2})) \\
&\quad \wedge (H', \neg p(-, v_2))\ \mathcal{AT}\ (H, p(v_1, v_2)) \\
&\quad \Rightarrow Attack(Ag_{WS_2}, Ag_{MWS_1_}C_x, (H', \neg p(-, v_2)))\big)
\end{aligned}$$

where

$$\begin{aligned}
&H = (Av_WS_2 + v_1 > Availability\text{-}C_x) \\
&\quad \wedge (v_2 > Participation\text{-}C_x) \text{ and} \\
&H' = Participation\text{-}C_y > v_2
\end{aligned}$$

4.3.4 Attack game

General form

The *Attack game* is specified as follows:

$$\begin{aligned}
&Attack(Ag_{WS_1}, Ag_{WS_2}, (H, p)) \wedge C_1 \\
&\quad \Rightarrow Accept(Ag_{WS_2}, Ag_{WS_1}, p) \\
&Attack(Ag_{WS_1}, Ag_{WS_2}, (H, p)) \wedge C_2 \\
&\quad \Rightarrow Challenge(Ag_{WS_2}, Ag_{WS_1}, H') \\
&Attack(Ag_{WS_1}, Ag_{WS_2}, (H, p)) \wedge C_3 \\
&\quad \Rightarrow Attack(Ag_{WS_2}, Ag_{WS_1}, (H', p')) \\
&Attack(Ag_{WS_1}, Ag_{WS_2}, (H, p)) \wedge C_4 \\
&\quad \Rightarrow Make\text{-}Offer(Ag_{WS_2}, Ag_{WS_1}, q) \\
&Attack(Ag_{WS_1}, Ag_{WS_2}, (H, p)) \wedge C_5 \\
&\quad \Rightarrow Refuse(Ag_{WS_2}, Ag_{WS_1}, p)
\end{aligned}$$

These conditions are identical to the ones associated with the *Justification game*.

An argumentative Web service Ag_{WS_2} accepts an attacker's argument if it can generate a support for it from its argumentation system. If it cannot neither generate nor negate this support, the agent challenges it. If it can generate a counter-attacker argument, then it will play the *Attack* move. If an offer can be made from the Web service's knowledge base using its argumentation system, it makes this offer. Otherwise, it refuses the attacker's argument. This *refuse move* can be played if the negation of the attacker's argument conclusion is in Ag_{WS_2}'s knowledge base. We note in this case that Ag_{WS_2} cannot play the *Attack move* since it does not have a counter-argument but only a knowledge about the negation of the argument conclusion.

Example 7 As a continuation of Example 6 in which, the Web service Ag_{WS_2} attacks the justification of the master $Ag_{MWS1\_}C_x$, this latter replies by making another offer $p(v_1, v_2 + \epsilon)$. In this new offer the master increases the promised participation rate by a value ϵ calculated using some rules from the argumentation system. Only the participation rate is increased since the availability increasing is not attacked:

$$\begin{aligned}
&\big(Attack(Ag_{WS_2}, Ag_{MWS_1_}C_x, (H', \neg p(-, v_2))) \\
&\quad \wedge v_2 + \epsilon \triangleleft Arg_Sys(Ag_{MWS_1_}C_x) \\
&\quad \Rightarrow Make\text{-}Offer(Ag_{MWS_1_}C_x, Ag_{WS_2}, p(v_1, v_2 + \epsilon))\big)
\end{aligned}$$

4.4 Dialogue games combination

Having specified the different dialogue games that argumentative Web services use in their interactions to manage their communities, we need to specify how these games could be now combined to form the persuasive negotiation protocol. We notice that during the same protocol session, an argumentative Web service cannot play the same move with the same content more than once. For example, if a master Web service proposed a participation rate by time unit v_2 during a protocol session, the same value cannot be proposed again during this session. Also, if an argumentative Web service uses a counter-argument to attack an argument, it cannot use the same counter-argument afterwards during this session (reiterations are prohibited). The protocol terminates (*Termination game*) either by accepting or refusing the last offer. There is an acceptance when a Web service accepts the offer (for example accepts the last offered rewards to join the community), i.e. when an agreement is reached. The protocol terminates by a refusal when no agreement is reached. The Persuasive Negotiation Protocol for Communities of Web Services ($\mathcal{PNP}$-$\mathcal{CWS}$) that combines the aforementioned games can be described using the BNF grammar as follows:

$$
\begin{aligned}
&\mathcal{PNP}-\mathcal{CWS} = \mathcal{E}ntry\ game\ ;\\
&\qquad\qquad (Refuse \mid (Accept\ ;\ ChG))\\
&ChG = Make\text{-}Offer\ ;\ X\\
&X = Accept\\
&\quad \mid Refuse\\
&\quad \mid Make\text{-}Offer\ ;\ X\\
&\quad \mid Attack\ ;\ X\\
&\quad \mid Challenge\ ;\ Justify\ ;\ X\\
&\quad \mid Attack\ ;\ Make\text{-}Offer\ ;\ X
\end{aligned}
$$

where "|" is the choice symbol, and ";" is the sequence symbol.
After the *Entry game*, the addressee refuses the invitation, or accepts to engage in negotiation, in which case chaining games (ChG) will take place. The last line in this grammar refers to the case where *Attack* and *Make-Offer* moves are played together.

5 Formal analysis

5.1 $\mathcal{PNP}$-$\mathcal{CWS}$'s properties

In this section, we discuss the computational and formal properties of the $\mathcal{PNP}$-$\mathcal{CWS}$ protocol. These properties are: *termination* (no deadlock), *soundness* (correct specification), and *completeness* (wholeness with respect to Web services' knowledge bases).

Proposition 1 *The $\mathcal{PNP}$-$\mathcal{CWS}$ protocol terminates iff the invited argumentative Web service refuses the $\mathcal{E}ntry$ game, or one of the Web services plays either Accept or Refuse moves when the $\mathcal{E}ntry$ game is accepted.*

Proof The first part is straightforward, because by definition, if the *$\mathcal{E}ntry$ game* is refused, the protocol terminates. Let us now suppose that the *$\mathcal{E}ntry$ game* is accepted. The direction $\Rightarrow$ is straightforward from the protocol's BNF description. In addition let us suppose that the protocol terminates. According to the protocol's BNF description, part X is recursive and all the moves, except *Accept* and *Refuse*, are followed by X. The only way to stop the process is then to play either *Accept* or *Refuse*. Consequently, the direction $\Leftarrow$ holds. □

Theorem 1 (Termination) *For any set of dialogue games, The $\mathcal{PNP}$-$\mathcal{CWS}$ protocol always terminates.*

Proof Because the knowledge bases of argumentative Web services are finite, the arguments that these Web services can build out of these bases are finite as well. Consequently, the number of offers and attacks that can be made and built respectively are finite. Therefore, the branches *Make-Offer* ; X, *Attack* ; X, and *Attack* ; *Make-Offer* ; X in the BNF description are finite, since playing the same move with the same content is prohibited during a conversation. The branch *Challenge* ; *Justify* ; X is also finite because the number of arguments is finite and when an argument is justified by itself, the addressee cannot challenge it again because repeating moves is prohibited. Thus in all possible executions, one of the argumentative Web services will select one of the branches *Accept* or *Refuse*. From Proposition 1 the result follows. □

Definition 5 (*Agreement*) Let Ag_{WS_1} and Ag_{WS_2} be two argumentative Web services engaged in a conversation using the $\mathcal{PNP}$-$\mathcal{CWS}$ protocol. Also, let $Arg\_Sys(Ag_{WS_1})$ and $Arg\_Sys(Ag_{WS_2})$ be their respective argumentation systems. An agreement about an offer p is reached between Ag_{WS_1} and Ag_{WS_2} iff $p \triangleleft Arg\_Sys(Ag_{WS_1})$ and $p \triangleleft Arg\_Sys(Ag_{WS_2})$.

In other words, an agrement about an offer is reached iff the offer can be supported by the the argumentation systems of the participating Web services.

Theorem 2 (Soundness) *If the $\mathcal{PNP}$-$\mathcal{CWS}$ protocol terminates by an acceptance* (*resp. refusal*)*, then an agreement is* (*rep. is not*) *reached.*

Proof According to the protocol's BNF description, an argumentative Web service plays *Accept* move as a reply to either an offer, an attack, or a justification. According to the *Offer game*, accepting an offer means that the addressee has an

argument in its knowledge base that supports accepting this offer. According to Definition 5, having this argument in the knowledge base means that an agreement is reached. Now, if the argumentative Web service accepts to either attack or justify, then according to the *Attack* and *Justification* games, the protocol's BNF description, and the fact that the content of an attack could be a counter-offer, this Web service accepts the last offer made by the addressee. Accepting this offer means that the Web service has an argument supporting it, which means that an agrement is reached.

In the opposite case, if an argumentative Web service plays a refusal, then according to the dialogue games specification and the protocol's BNF description, all the exchanged offers can not be supported by one of the two Web services. This means that there is no argument from the two Web services' knowledge bases supporting one of the offers. Consequently, an agreement is not reached. □

The soundness property shows that the protocol is correct. However, what is important here to show is that if more than one agreements are made available for argumentative Web services, then the protocol execution will reach one of them.

Theorem 3 (Completeness) *If an agreement about an offer p can be reached from the knowledge bases of the argumentative Web services, then the protocol execution will result in achieving an agreement.*

Proof According to Definition 5, the existence of an agreement about p means that $p \triangleleft Arg\_Sys(Ag_{WS_1}) \cup Arg\_Sys(Ag_{WS_1})$. Then, from the union of the two knowledge bases, it is possible to build an argument supporting the offer p, which is not attacked by another argument from the union. Let us show how this argument can be reached when executing the protocol.

If p is the initial offer made by Ag_{WS_1} for example, then Ag_{WS_2} will accept it since $p \triangleleft Arg\_Sys(Ag_{WS_2})$. So an agreement is reached. If the initial offer is q, we have $q \simeq p$ since p and q are different but about the same topic. According to Proposition 1, the protocol terminates by either a refusal or an acceptance. Because the protocol always terminates by Theorem 1, during the protocol execution one of the argumentative Web services should play either *Refuse* or *Accept* move. Suppose that *Refuse move* is played by one of the two Web services, for example Ag_{WS_1}. According to the dialogue games specification, there is no possibility for this Web service to make a counter-offer r such that $r \simeq q \wedge r \triangleleft Arg\_Sys(Ag_{WS_1})$. This is contradictory because by hypothesis there is an offer p such that $p \simeq q \wedge p \triangleleft Arg\_Sys(Ag_{WS_1})$. Consequently, the only possibility to terminate the protocol is to play an acceptance move, which means that an agreement is reached. □

It is also possible to use a *proof by construction* to prove this theorem by discussing all possible situations according to the protocol's BNF description. We let this exercise to interested readers.

We notice here that the soundness theorem states that an agreement is reached, but does not tell what is the agreement. The reason is that many agreements can exist, and which one could be reached depends on the strategies that these argumentative Web services adopt.

5.2 Complexity

In this section, we discuss the complexity of the $\mathcal{PNP}$-$\mathcal{CWS}$ protocol. Because first, the protocol is expressed in terms of argumentation-based dialogue games, and second, the decision parameters (the conditions associated with the rules) that argumentative Web services use to combine these games are expressed in terms of the possibility of building arguments, the complexity of the protocol is determined by the complexity of generating arguments to support offers or to attack existing arguments. In the following we present the different complexity results.

Proposition 2 *Given a Horn knowledge base Γ, a subset $H \subseteq \Gamma$, and a formula h. Checking whether (H, h) is an argument is polynomial.*

Proof From the linear time algorithms for Horn satisfiability in [14], it follows that the Horn implication problem $H \vdash h$ is decidable in $O(|H| \times |h|)$ time. From the same result, it also follows that deciding whether H is consistent is polynomial. □

Proposition 3 *Given a Horn knowledge base Γ, and an argument (H, h) over Γ. Checking whether (H, h) is minimal is polynomial.*

Proof Let l be a literal. The following algorithm resolves the problem:
$\forall l \in H$ check if $H - \{l\} \vdash h$. Because the implication problem is polynomial, we are done. □

As indicated in Sect. 4.1, argumentative Web services are equipped with knowledge bases that are supposed to be consistent. Let us consider this case.

Proposition 4 *Let Γ be a definite Horn knowledge base, h a formula, and $\mathcal{A}$ the set of arguments over Γ.*
$\exists H \subseteq \Gamma : (H, h) \in \mathcal{A} \Rightarrow \forall H' : H \subseteq H' \subseteq \Gamma,\ (H', h) \in \mathcal{A}$.

Proof If (H, h) is an argument where H is a set of definite Horn formulas under the form c or $p_1 \wedge p_2 \wedge \cdots \wedge p_n \rightarrow c$ where $p_1, p_2, \ldots, p_n, c$ are positive literals, then adding any definite Horn formula to H will result in a consistent set of formulas $H' : \Gamma \supseteq H' \supseteq H$. Since $H \vdash h$, it follows that $H' \vdash h$, whence the proposition. □

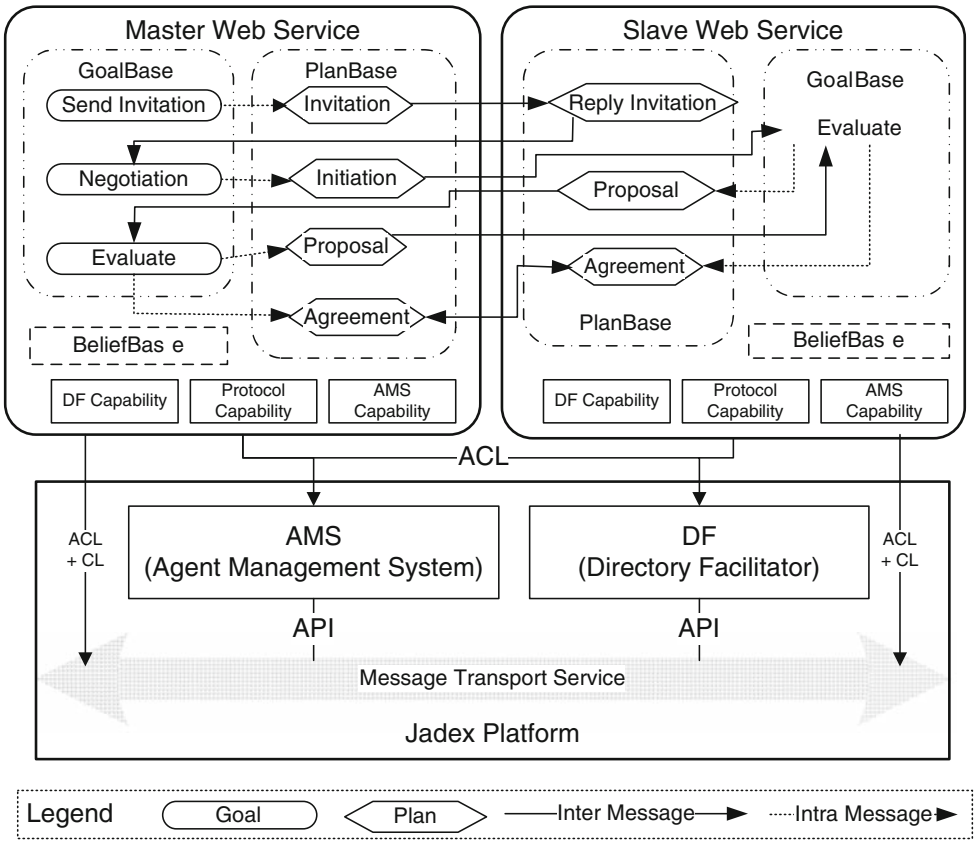

Fig. 3 Architecture of the prototype

Fig. 4 Parameters of the *Proposal* plan

```
<plans>
      ……
      <plan name="proposal">
            <parameter name="cfp" class="Object">
               <goalmapping ref="evaluate_proposals.cfp"/>
            </parameter>
            <parameter name="cfp" class="Object" optional="true">
               <goalmapping ref="evaluate_proposals.cfp"/>
            </parameter>
            <parameterset name="proposals" class="Object">
               <goalmapping ref="evaluate_proposals.proposals"/>
            </parameterset>
            <parameterset name="history" class="Negotiation">
               <goalmapping ref="evaluate_proposals.history"/>
            </parameterset>
            <parameterset name="acceptables" class="Object">
               <goalmapping>
                           ref="evaluate_proposals.acceptables"
               <goalmapping/>
            </parameterset>
            <body class="EvaluatePlan" />
            <trigger>
               <goal ref="evaluate_proposals"/>
            </trigger>
      </plan>
            ……
</plans>
```

Theorem 4 *Given a definite Horn knowledge base Γ and a formula h. Deciding whether there is an argument (H, h) is polynomial.*

Proof From Proposition 4, it follows that there is an argument supporting h iff $(\Gamma, h) \in \mathcal{A}$. Because every definite Horn knowledge base is a Horn knowledge base, then by Proposition 2, the theorem follows. □

The following theorem is a direct consequence of Theorem 4.

Theorem 5 *Given a consistent Horn knowledge base Γ and a formula h. Deciding whether there is an argument (H, h) is polynomial.*

Proof Proposition 4 holds if the knowledge base Γ is consistent. Then, by Proposition 2, the result follows. □

Proposition 5 *Let Γ be a Horn knowledge base and (H, h) and $(H'.h')$ be two arguments over Γ. Deciding whether (H', h') $\mathcal{AT}$ (H, h) is polynomial.*

Proof According to Definition 2, (H', h') $\mathcal{AT}$ (H, h) iff $H' \vdash \neg h$. The proof is then straightforward since the Horn implication problem $H' \vdash \neg h$ is decidable in $O(|H| \times |\neg h'|)$ time. □

Theorem 6 *Let Γ be a consistent Horn knowledge base and (H, h) an argument over Γ. Deciding whether there is an attacker of (H, h) over Γ is polynomial.*

Proof From Definitions 1 and 2, building an argument attacking a given argument is less complex than building an argument supporting a conclusion. From Theorem 4 we are done. □

These results prove that our $\mathcal{PNP}$-$\mathcal{CWS}$ protocol is computationally efficient, and its complexity depends only on the size of the knowledge bases.

6 Implementation

6.1 Architecture

A prototype has been implemented to demonstrate first, the combination between argumentative agents and Web services and second, the performance of the persuasive negotiation protocol $\mathcal{PNP}$-$\mathcal{CWS}$ that manages communities of Web services. The prototype is built on top of Jadex platform [40]. Based on Java and XML technologies, Jadex allows building up rational and goal-oriented agents.

Figure 3 depicts the architecture of the prototype. Argumentative Web services have capabilities that represent their beliefs, goals, plans, and events. These capabilities are specified in an XML-based *Definition File*. The description of the Web service and its non-functional parameters (i.e. QoS) are examples of beliefs. The master Web service has three goals that populate its *goalbase*: (1) *Send Invitation* used to send joining invitations; (2) *Negotiation* used to trigger the negotiation of the joining contracts; and (3) *Evaluate* used to trigger the evaluation of the different offers it receives back. Four plans in the master Web service are associated to these goals: (1) *Invitation* used to process the joining invitation; (2) *Initiation* triggered by the *Negotiation* goal; (3) *Proposal* to make and evaluate offers; and (4) *Agreement* to stop the negotiation either by an agreement or not. The slave Web service has three plans: (1) *Reply Invitation* used to reply to the joining invitation; (2) *Proposal* to make offers; and (3) *Agreement* to finish the negotiation. *Evaluate* goal in the slave Web service is used to trigger the *Proposal* and *Agreement* plans.

Each goal has a target condition and is activated when this condition evaluates to true. For example, *Negotiation* goal in the master Web service is activated when an acceptance reply is received by a slave Web service. When a plan is triggered, the Web service runs it to perform the specified actions. For example, when *Evaluate* goal is activated, the plan *Proposal* is executed to evaluate the received offer and/or make a new offer. The different parameters of a plan are specified in an XML file. Figure 4 gives an example of the different parameters used in the *Proposal* plan. For example the file "evaluate_proposals.cfp" refers to the mapping goal file of this plan.

The argumentation reasoning is implemented in Java as the procedural parts of the plans. The $\mathcal{PNP}$-$\mathcal{CWS}$ protocol is declaratively specified as a set of dialogue games using an XML syntax in a public plan that argumentative Web services refer to when communicating. Which dialogue game a Web service should play within this protocol depends on its strategy based on its argumentation system and the message it receives from the addressee. The exchanged messages are events that trigger some plans. Web services share the same ontology to define the meaning of the Horn formulas they exchange. The whole prototype relies on the Jadex platform, which provides a directory facilitator, an agent management system, and a message transport service. The message transport service is provided to transport messages between Web services by specifying the recipient of a message, the general structure of the message, and the used ontology. The agent management system facilitates the creation, registration, location and operation of Web services. All the simulated Web services and communities are registered in the directory facilitator. The directory facilitator used to report Web services in a community is controlled by the master Web service. When a Web service accepts a joining invitation, the master adds it to its directory. When the master decides to put

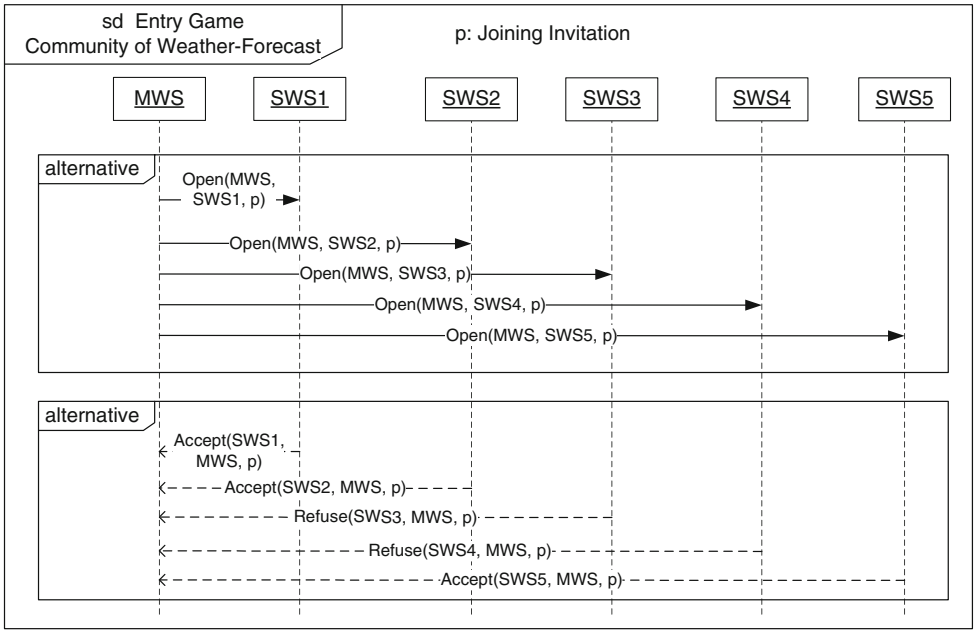

Fig. 5 Sequence diagram of the *Entry game* scenario

a Web service out of the community, it deletes its registration information from the directory.

6.2 Experimental results

To illustrate some experimental results, we consider two scenarios: (1) invitation to join a community; and (2) negotiation of the joining contract. In the first scenario, a Master Web service MWS searches for slave Web services providing `Weather-Forecast` functionality in the directory facilitator. Five Slave Web Services are identified $SWS_{i,i=1,\ldots,5}$, and the master plays the *Entry game* (*Open* communicative act) to invite them to negotiate about joining its community. Three of these Slave Web services (SWS_1, SWS_2, and SWS_5) accept the game because they have arguments supporting the acceptance decision. These arguments are `availability` (these Web services did not commit to join any other community yet) and `interest` in joining the community of this master Web service because they believe that its reputation is high. The rest of the slave Web services namely SWS_3 and SWS_4 refuse the invitation for two different reasons. SWS_3 is already committed to join another community, and SWS_4 is not interested because it has an argument against the reputation of this community. Figure 5 illustrates the sequence diagram of this scenario. According to the *Entry game*, slave Web services do not reveal their arguments about their acceptance or refusal, but they use them internally to make their decisions. This is the reason why they are not illustrated in the sequence diagram. Figure 6 illustrates a snapshot of this scenario.

In the second scenario, the master Web service negotiates the joining contract with one of the slave Web services that accept the joining invitation, for instance SWS_1. As in Example 5 of Sect. 4.3.2, two terms are negotiated in this scenario: SWS_1's `availability` increase and `participation rate` in composite business scenarios the master Web service can provide. First, the master Web service uses its argumentation system to make an offer: `promised participation rate` by time unit = 20% and `availability` increase by time unit = 30% (these values are calculated on the basis of the current efficiency of the community and what the master provides to the existing members). By playing the *Offer game*, SWS_1 attacks the first part of the offer because it has an argument against this offer; the proposed `participation rate` is less than the community average (we call this argument the average argument). The master Web service replies, using the *Attack game*. It makes a new offer in which the `participation rate` is increased (from 20 to 30%) and the `availability` is decreased (from 30 to 25%). Playing the *Offer game*, $SWS1$ replies to this new offer by attacking the second part using the argument that adding its current availability to the offered rate is less than the community average. The master makes then a new offer by increasing the `availability rate` (from 25 to 35%) but by reducing the `participation` rate (from 30 to 25%). This new offer considers the fact that it is greater than the community average, so it cannot be attacked using the average argument. SWS_1 replies to this new offer by making another counter-offer using its argumentation system. In this offer, SWS_1 requests a 28% for the `participation rate` instead of 25% and accepts the proposed `availability` increase. The master Web service proposes then a balance between the two values: 27% for the `participation rate`, and 32% for the

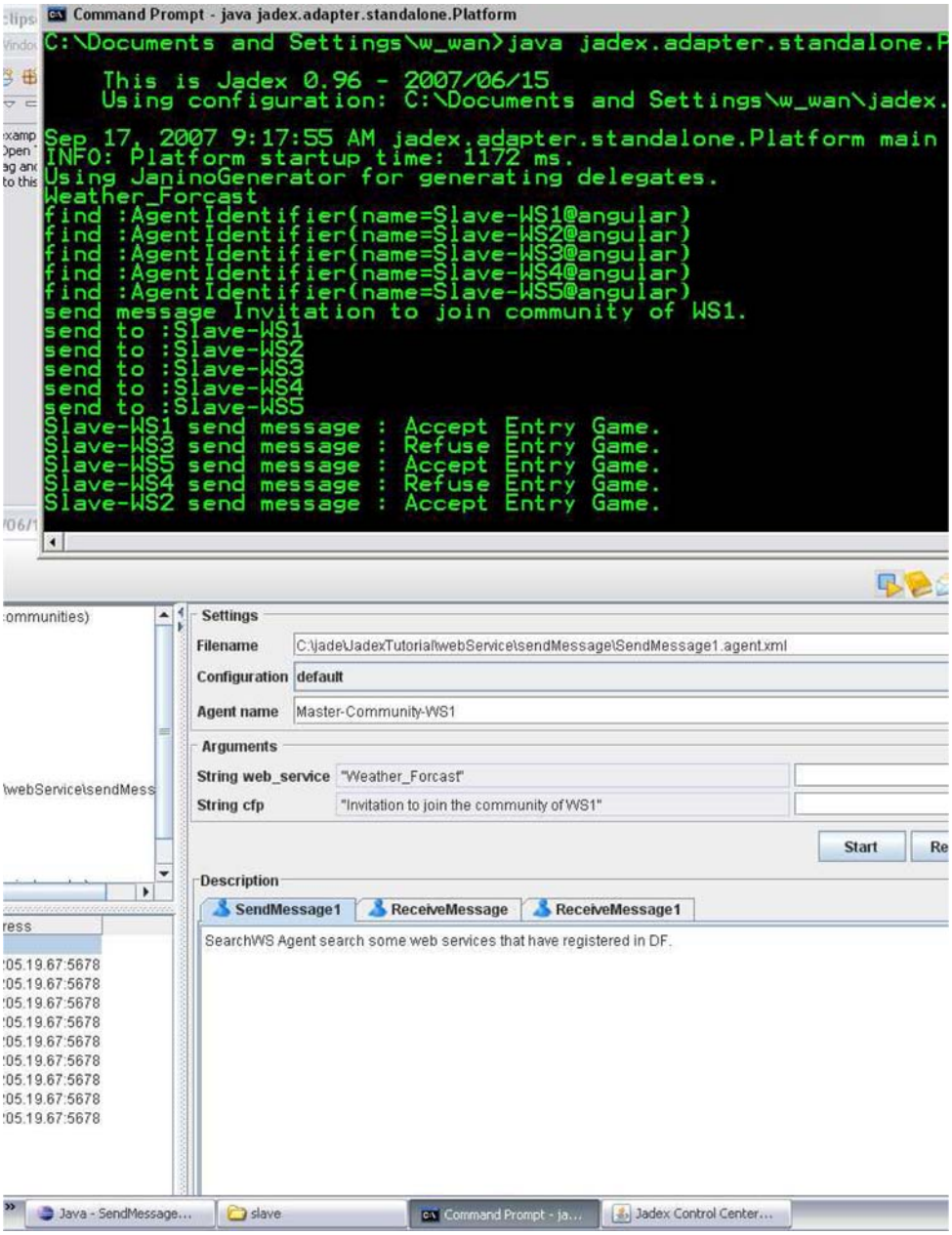

Fig. 6 Snapshot for the *Entry game* scenario

availability increase. Finally, SWS_1 accepts the new offer, which results in stopping the negotiation. This scenario is illustrated with Fig. 7.

6.3 Discussions

The two implemented scenarios explained in the experimental results (inviting Web services to join a community and negotiating a joining contract) show how argumentative agents are combined with Web services to effectively manage communities that host these Web services. In fact, these results show how the problems discussed in this paper (starvation, competition-free, and unfairness) are resolved in a satisfactory manner. Based on their argumentation systems, some slave Web services have accepted joining the community while some have refused. In addition, not all the Web services that have accepted the invitation were engaged in negotiation, but only few depending on the needs of and resources available to the master Web service. In addition, by negotiating the joining contract, SWS_1 has succeeded to obtain a better offer compared to the one that has been proposed by the master (27% in stead of 20% of participation rate and 32% instead of 30% of availability increase). Consequently, starvation,

non-competitiveness and unfairness are properly addressed in a community of Web services. Another outcome of the implementation exercise concerns the correctness and high performance of the $\mathcal{PNP}$-$\mathcal{CWS}$ protocol. The answers given by the invited Web services in the invitation session of the protocol are compatible with their knowledge bases. Only the Web services that can theoretically accept the invitation (regarding the content of their knowledge base) send an acceptance. In term of performance, the whole invitation protocol is executed in very few seconds (the carried out simulations have exhibited an execution time between 2 and 7 s with a number of Web services ranging from 5 to 30). The negotiation sessions in the experimental results have also shown the soundness and termination of the protocol. An agreement is reached only if it is possible regarding the content of the participants' knowledge bases. In terms of execution time, the whole protocol is linear with the size of the knowledge bases and generally takes between 4 and 8s when this size ranges from 50 to 100 propositions.

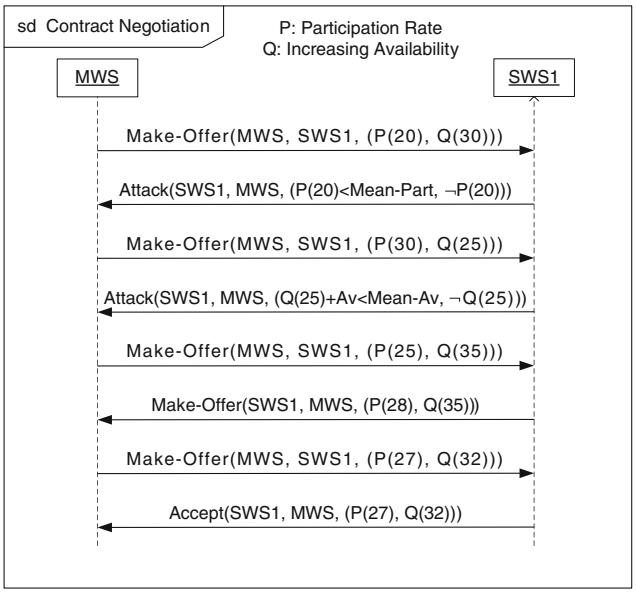

Fig. 7 Sequence diagram of the contract negotiation scenario

7 Conclusion

In this paper, we first, exposed some obstacles that restrict the use of Web services in complex business applications and then, offered some solutions to tackle these obstacles through two concepts namely community and argumentative software agents. On the one hand, community first, groups Web services with the same functionality regardless of who developed them, where they are located, and how they function and second, sustains Web services availability at run-time. On the other hand, argumentative agents back Web services in making appropriate decisions prior they implement any action. Different types of actions were listed like signing up in a community, participating in a composition scenario, and last but not least leaving a community due to lack of business opportunities.

We discussed the rationale behind using argumentation to manage Web services communities, and we showed that this technique provides suitable solutions for some problems, which we denoted by "starvation," "non-competitiveness," "unfairness," and "selection" that cannot be efficiently resolved using Web services' existing techniques. We framed the management operations that take place in a community with a persuasive negotiation protocol that argumentative agents implement. This protocol relies on a set of games like $\mathcal{E}ntry$, $\mathcal{O}ffer$, and $\mathcal{A}ttack$. The formal properties of this protocol along with its complexity were discussed and assessed, respectively. In addition, a prototype simulating the protocol was discussed. The experimental results revealed that inviting Web services to join communities and negotiating joining contracts with multi-issue terms can be efficiently managed using argumentation. In addition, the implementation allowed us to experimentally check the theoretical soundness and completeness of the proposed protocol.

Our research work on argumentative agents and communities of Web services opens up at least two research opportunities namely alliance development and reputation. Firstly, alliances could be used to internally structure a community based on some mutual agreements between providers of Web services as part of their partnership strategies. Secondly, reputation would enable the existence of several communities of Web services. This existence would need to be looked into from two perspectives: provider and user. In the former perspective, a provider should primarily know in which community it will let its Web services sign up. In the latter perspective, a user should know which community to bind to prior to identifying the Web service (that resides in this community) to invoke later.

Acknowledgments The authors would like to thank the reviewers for their very valuable comments and suggestions of changes. The first author acknowledges the financial support of NSERC, FQRNT, and FQRSC (Canada).

References

1. Baldoni M, Baroglio C, Martelli A, Patti V (2007) Reasoning about interaction protocols for customizing web service selection and composition. J Logic Algebraic Program (special issue on Web Services and Formal Methods) 70(1)
2. Barbir A (2003) Web services security: an enabler of semantic Web services. In: Proceedings of business agents and the semantic Web held in conjunction with The 16th Canadian conference on artificial intelligence (AI'2003). Halifax, Nova Scotia
3. Benatallah B, Casati F, Toumani F (2005) Representing, analysing and managing Web service protocols. Data Knowl Eng J 58(3)

4. Benatallah B, Dumas M, Sheng QZ (2005) Facilitating the rapid development and scalable orchestration of composite Web services. J Distrib Parallel Databases 17
5. Bentahar J, Labban J, Moulin B (2007) An argumentation-driven model for efficient and secure negotiation. In: Proceedings of the international conference on group decision and negotiation (GDN2007), Montreal
6. Bentahar J, Maamar Z, Benslimane D, Thiran P (2007) An argumentation framework for communities of Web services. IEEE Intell Syst 22(2):6
7. Birman KP (2004) Like it or not, Web services are distributed objects. Commun ACM 47(12)
8. Bui T, Gacher A (2005) Web services for negotiation and bargaining in electronic markets: design requirements and implementation framework. In: Proceedings of the 38th Hawaii international conference on system sciences (HICSS'2005), Big Island
9. Chang S, Chen Q, Hsu M (2003) Managing security policy in a large distributed Web services environment. In: Proceedings of the 27th annual international computer software and applications conference (COMPSAC'2003), Dallas
10. Chesñevar CI, Maguitman A, Loui R (2000) Logical models of argument. ACM Comput Surv 32
11. Dale J, Ceccaroni L, Zou Y, Agam A (2003) Implementing agent-based Web services. In: Proceedings of the AAMAS'03 workshop on challenges in open agent systems, Melbourne
12. Daniel F, Pernici B (2005) Insights into Web service orchestration and choreography. Int J E-Bus Res (The Idea Group Inc.) 1(2)
13. Dinis L, Parrondo JMR (2004) Inefficiency of voting in parrondo games. Phys A Stat Mech Appl 343
14. Dowling W, Gallier JH (1984) Linear-time algorithms for testing the satisfiability of propositional horn theories. J Logic Program 1(3)
15. Dung PM (1995) On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif Intell 77(2)
16. Dung PM, Kowalski RA, Toni F (2006) Dialectic proof procedures for assumption-based, admissible argumentation. Artif Intell 170(2)
17. Elnaffar S, Maamar Z, Yahyaoui H, Bentahar J, Thiran P (2008) Reputation of communities of web services—preliminary investigation. In: Proceedings of the international symposium on Web and mobile information services (WAMIS'2008) held in conjunction of the 22nd international conference on advanced information networking and applications (AINA'2008), Okinawa
18. Elvang-Goransson M, Fox J, Krause P (1993) Dialectic reasoning with inconsistent information. In: Proceedings of the 9th conference on uncertainty in artificial intelligence (UAI'1993), Washington, DC
19. Fensel D (2001) Ontologies: a silver bullet for knowledge management and electronic commerce. Springer, Heidelberg
20. Jennings N, Sycara K, Wooldridge M (1998) A roadmap of agent research and development. Autonomous Agents and Multi-Agent Systems, Kluwer, vol 1(1)
21. Jureta I, Faulkner S, Achbany Y, Saerens M (2007) Dynamic Web service composition within a service-oriented architecture. In: Proceedings of the IEEE international conference on Web services (ICWS'2007), Salt Lake City
22. Li Y, Shen W, Chenniwa H (2004) Agent-based Web services framework and development environment. Comput Intell 20(4)
23. Ma KJ (2005) Web services: what's real and what's not. IEEE IT Professional, vol 7(2)
24. Maamar Z, Benslimane D, Mostéfaoui GK, Subramanian S, Mahmoud Q (2008) Towards behavioral Web services using policies. IEEE Trans Syst Man Cybern A (forthcoming)
25. Maamar Z, Benslimane D, Narendra NC (2006) What can context do for Web services? Commun ACM 49(12)
26. Maamar Z, Lahkim M, Benslimane D, Thiran P (2006) Towards an approach for specifying and managing communities of Web services. Technical report, Zayed University, King Saud University, Claude Bernard Lyon 1 University, and Namur University
27. Maamar Z, Lahkim M, Benslimane D, Thiran P, Sattanathan S (2007) Web services communities—concepts and operations. In: Proceedings of the 3rd international conference on Web information systems and technologies (WEBIST'2007), Barcelona
28. Maximilien EM, Singh M (2005) Toward Web services interaction styles. In: Proceedings of IEEE services computing conference (SCC2005), Orlando
29. McBurney P, Parsons S (2002) Games that agents play: a formal framework for dialogues between autonomous agents. J Logic Lang Inform 11(3)
30. Medjahed B, Atif Y (2007) Context-based matching for Web service composition. Distrib Parallel Databases. Springer, Heidelberg 21(1)
31. Medjahed B, Bouguettaya A (2005) A dynamic foundational architecture for semantic Web services. Distributed and Parallel Databases. Kluwer, Dordrecht, vol 17(2)
32. Menascé DA (2002) QoS issues in Web services. IEEE Internet Comput 6(6)
33. Meyer J-J, Veltman F (2007) Inteligent agents and common sense reasoning. In: Blackburn P et al (ed) Studies in logic and practical reasoning. Handbook of Modal Logic, vol 3
34. Mrissa M, Ghedira C, Benslimane D, Maamar Z, Rosenberg F, Dustdar S (2007) A context-based mediation approach to compose semantic Web services. ACM Transactions on Internet Technology, Special Issue on Semantic Web Services: Issues, Solutions and Applications, vol 8(1)
35. Narendra NC (2001) Flexible agent societies: flexible workflow support for agent societies. In: Proceedings of the 2001 international conference on intelligent agents Web technologies and Internet commerce (IAWTIC'2001), Las Vegas
36. Ouzzani M, Bouguettaya A (2004) Efficient access to Web services. IEEE Internet Comput 8(2)
37. Paolucci M, Sycara K (2003) Autonomous semantic Web services. IEEE Internet Comput 7(5)
38. Parsons S, Wooldridge M, Amgoud L (2003) Properties and complexity of some formal inter-agent dialogues. J Logic Comput 13(3)
39. Pitt J, Kamara L, Sergot MJ, Artikis A (2005) Formalization of a voting protocol for virtual organizations. In: Proceedings of the international conference on autonomous agents and multi-agent systems (AAMAS'2005), Utrecht
40. Pokahr A, Braubach L, Lamersdor W (2005) Jadex: a BDI reasoning engine. In: Bordini R, Dastani M, Dix J, Seghrouchni A (eds) Multi-agent programming. Languages, Plateforms and Applications. Springer, Heidelberg
41. Prakken H, Vreeswijk G (2000) Logics for defeasible argumentation, 2nd edn. Handbook of Philosophical Logic
42. Smith R (1980) The contract net protocol: high level communication and control in distributed problem solver. IEEE Trans Comput 29
43. Taher Y, Benslimane D, Fauvet M-C, Maamar Z (2006) Towards an approach for Web services substitution. In: Proceedings of The 10th international database engineering and applications symposium (IDEAS'2006), Delhi
44. Toni F, Bentahar J (2008) Computational logic-based agents. J Auton Agents Multi-Agent Syst 16(3)
45. Wanyamaa T, Homayoun B (2007) A protocol for multi-agent negotiation in a group-choice decision making process. J Network Comput Appl 30(3)