Intelligent IEEE Systems
www.computer.org/intelligent

An Argumentation Framework for Communities of Web Services

Jamal Bentahar, *Concordia University, Montreal*

Zakaria Maamar, *Zayed University*

Djamal Benslimane, *Claude Bernard Lyon 1 University, France*

Philippe Thiran, *University of Namur*

Vol. 22, No. 6
November/December 2007

An Argumentation Framework for Communities of Web Services

Jamal Bentahar, *Concordia University, Canada*

Zakaria Maamar, *Zayed University*

Djamal Benslimane, *Claude Bernard Lyon 1 University, France*

Philippe Thiran, *University of Namur*

Argumentation theory, implemented through a set of software agents that reason about Web services, can improve Web services' performance through the notion of communities.

As the number of Web services continues to increase, so does the opportunities to compose them to build more complex and complete business solutions. To facilitate and speed up Web-services discovery, Web services with similar (or equivalent) functionalities—such as flight booking and travel reservation—can be grouped into *communities*.[1]

Consequently, the multiagent research community views Web services as a substantial application domain. However, so far, the use of agents-based Web services has been confined to simple software entities with basic interaction and decision-making capabilities. Here, we discuss how to enrich agents to apply logic-based reasoning and argumentation.

Argumentation is a dialectical process that uses pro and con arguments to reach a conclusion. Arguments interact by attacking each other through an attack binary relation. In agent-based computing, argumentation can help agents interact rationally by letting them give reasons that support their conclusions and receive counterarguments. Simply put, an argumentative agent employs a dialectical process when it wants to affirm or disavow the conclusions it's conveying to peers.[2] Using these agents, communities of Web services can manage themselves and argue with peers about their status and the status of their respective communities. We propose an argumentation framework that defines interaction mechanisms for peers in these communities.

Communities of Web services

Although Web services in a community have a common functionality, they can have distinct nonfunctional properties.[3] Additionally, a community can describe a desired functionality without explicitly referring to any concrete (or preselected) Web service that will implement this functionality at runtime.[1]

Figure 1 represents an argumentative agent-based environment consisting of Web services communities; Web service providers; Universal Description, Discovery, and Integration (UDDI) registries; and software agents. Even after a Web service joins a community,

- the process for describing, announcing, and invoking Web services remains the same;
- the services that UDDI registries regularly offer, such as announcement and consultation, remain

1541-1672/07/$25.00 © 2007 IEEE
Published by the IEEE Computer Society

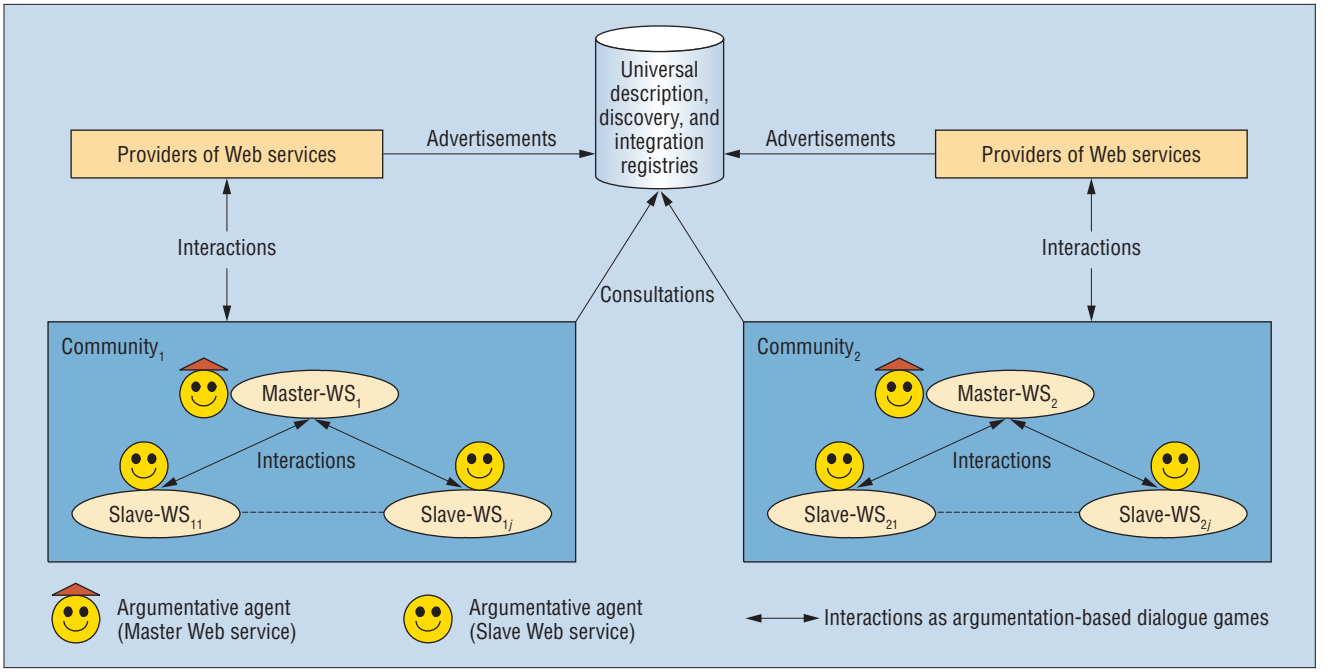

Figure 1. Web services, communities, and argumentative agents.

the same; and
- the selection process is transparent, so users don't have to know that the Web services are gathered into communities.

A *master* Web service leads a community using special services related to community management, such as attracting and retaining services, which its respective *argumentative master agent* offers. It can also act as a broker, matching users' needs with Web services' functionalities.[1] All other Web services in a community are denoted as *slaves* (associated with *argumentative slave agents*). A master Web service can be designated in one of two ways. A community designer can designate a Web service as the master for a particular community (the approach we adopt), or a slave Web service can be selected from the list of slave Web services that already populate a community. In the latter, the selection could happen on a voluntary basis (a functionality the designer allows) or after the agents or community designer run an election among the slave Web services.

In figure 1, Web services coupled with agents engage peers in conversations in which the agents aim to persuade or negotiate with each other. These conversations combine argumentation-based dialogue games. The master Web service can persuade a Web service to join or remain in its community. Furthermore, in the same community, slave Web services can mutually sort out their participation in composite Web services when conflicts arise.

Managing a community of Web services involves three main activities: developing the community (or dismantling it, if need be), attracting new Web services, and retaining existing Web services.

Community development and dismantlement

Establishing a community is a designer-driven activity that occurs in two steps. First, a community designer defines a community's functionality, such as flight booking, using, for example, a dedicated ontology. Then the designer deploys the master Web service, which invites Web services to join the community. A community's survival, to a certain extent, depends on how much the slave Web services participate in compositions.

Dismantling a community is also a designer-driven activity. The master Web service monitors the arrival of new Web services and departure of existing ones. It also identifies Web services to be part of composite Web services and sanctions misbehaving Web services. If the number of Web services in the community is less than a certain threshold, and the number of participation requests in composite Web services over some time period is less than another threshold, the master Web service will dismantle the community. The designer sets both thresholds. A Web service ejected from one community can be invited to join other communities, as long as it offers a similar functionality.

Web services attraction and retention

The master Web service regularly interacts with UDDI registries to stay informed of their Web services advertisements, and it can use rewards to attract new Web services. Additionally, a new Web service can approach a master Web service if it's interested in being part of its community. The new Web service can try to persuade the master Web service through argumentation that it will bring additional capabilities (such as better nonfunctional properties) to the community. The master Web service should also nominate (as part of its broker role) slave Web services to participate as components in composite Web services. To this end, the master Web service uses the contract-net protocol, which consists of asking slave Web services to bid on certain participation activities in compositions.

The ability to retain Web services in a community indicates two things. First, although a community's Web services must compete with each other, it indicates that they also exhibit a cooperative attitude. Second, it indicates that a Web service (owner) is, to a certain extent, sat-

isfied with its level of participation in composite Web services.

A master Web service can also ask a Web service to leave a community. It might request this if a Web service has a new functionality that doesn't perfectly match the community's functionality. Or it could do so if the service has been unreliable, failing to participate in composite Web services owing to recurrent operation problems.

Toward argumentative Web services

Formally, an argumentation system comprises a *logical language* £ and defines *arguments*, *attack relations* between arguments, and terms of *acceptability*. Several proposed argumentation theories and frameworks implement argumentation systems.[4,5] Argumentation systems based on propositional logic are intractable[6] and thus not suitable for Web services and their required business scenarios. Argumentation systems based on logic programming lack the logical inference required for the advanced argumentation mechanisms involved in typical business negotiations.

We've found that using a restricted language such as *propositional Horn clauses* is sufficient to represent and reason about knowledge that Web services use during argumentative conversations. A propositional Horn clause is a disjunction of literals (an elementary or atomic proposition or its negation) with at most one positive literal $\neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_n \vee c$ (also written as implication $p_1 \wedge p_2 \wedge \ldots \wedge p_n \rightarrow c$). A propositional Horn formula is a conjunction of propositional Horn clauses. We focus on a further restriction called *propositional definite Horn clauses*, 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 particular interest in modeling argumentation reasoning, because formulas of the type $p_1 \wedge p_2 \wedge \ldots \wedge p_n \rightarrow c$ are adequate to describe interrelationships between premises and conclusions. Agents could use this to support positive literals.

> Argumentation systems based on logic programming lack the logical inference required for the advanced argumentation mechanisms involved in typical business negotiations.

Web services can use these formulas together with a logical inference to reason and argue about different matters. For example, to join a community, an interested Web service could, through its agent, persuade a master Web service by presenting the conditions that this master uses (literals $p_1, p_2, \ldots, p_n$) as premises of the conclusion c, which represents the fact of joining the community. For this Web service, these premises are true, so conclusion c is logically inferred. If not, this Web service should be able to present propositional definite Horn formulas supporting the premises $p_1, p_2, \ldots, p_n$. For example, assume that a Web service, through its agent, believes that $p_{11}, p_{12}, \ldots, p_{1n}$ are true. Then, it can present the following argument supporting p_1: $p_{11} \wedge p_{12} \wedge \ldots \wedge p_{1n} \rightarrow p_1$.

Furthermore, a master Web service can persuade a new Web service to join a community by presenting the benefits of being a member. One excellent persuasive argument that the master Web service could use is the fact that the participation rate in compositions is high in a community. A propositional definite Horn formula could express such an argument.

In addition, Web services can use (general) propositional Horn logic and the logical inference to build arguments attacking the addressees' arguments. For example, a new Web service can attack a master Web service's persuasive argument about the high member-participation rate in composite Web services, arguing that some members' participation rate isn't as claimed. Also, a master Web service can attack a slave's argument about its ability to join the community—for example, if the slave's QoS is less than the community average. Peers that have a much higher quality of service could also attack this slave's argument.

Our argumentation framework differs from the one based on logic programming. Propositional Horn formulas aren't inference rules in the sense of logic programming but are propositional formulas with material implication. Consequently, agent-based Web services could use logical inference to infer new arguments for or against propositions from these knowledge bases.

In the following, we define the concepts the argumentation framework uses to manage communities of Web services. We also discuss the framework's underlying computational complexity. Note that $\vdash$ stands for classical inference and, for the sake of simplicity, we omit the word "propositional" (so, "definite Horn theory" designates "propositional definite Horn theory").

Definition 1 (argument). *Let* Γ *be a consistent knowledge base with no deductive closure under the form of a Horn theory (that is, contains only Horn formulas). A Horn argument is a pair* (H, h) *where* h *is a formula and* H *a subset of* Γ *such that (i)* H *is consistent and (ii)* $H \vdash h$. *The set* H *is the argument's* support *and* h *its* conclusion.

In some frameworks, H is minimal, so no proper subset of H satisfying both (i) and (ii) exists. We exclude this strong condition because it increases the polynomial hierarchy's complexity by one level.[6] However, agent-based Web services can use nonminimal Horn arguments (or arguments, for the sake of simplicity). If a premise gets attacked, the agent generates a conclusion to test the attacked premise's relevancy. Generating a conclusion from a set of nonattacked premises means that the attacked premise is irrelevant to the conclusion.

Definition 2 (relevant premise). *Let* (H, h) *be an argument and* $x \in H$ *be a premise.* x *is relevant to* (H, h) *if and only if* $H - \{x\} \nvdash h$.

Proposition 1. *Let* (H, h) *be an argument.* (H, h) *is minimal if and only if* $\forall x \in H$, x *is relevant to* (H, h).

Proof. *The proof is straightforward from the definitions.*

Definition 3 (attack). $\mathcal{AT}$ *is a binary relation between arguments. Let* Γ_1 *and* Γ_2 *be two consistent Horn knowledge bases (that is, they contain a consistent set of Horn clauses). Let* (H, h) *and* (H', h') *be two arguments over* Γ_1 *and* Γ_2, *respectively.* (H', h') $\mathcal{AT}$ (H, h) *if and only if* $H' \vdash \neg h$ *or* $\exists x \in H : H' \vdash \neg x$ *and* x *is relevant to* (H, h) *where "¬" represents strict negation. In other words, an argument is attacked if and only if another argument exists that strictly negates its conclusion or one of its relevant premises.*

Example 1. *Let* $\Gamma_1 = \{p_1, p_2, p_3, p_1 \wedge p_2 \wedge p_3 \rightarrow c, r, s, r \wedge s \rightarrow t\}$ *and*

$\Gamma_2 = \{\neg q_1 \rightarrow \neg c, m, n, \neg q_1\}$. $((p_1, p_2, p_3), c)$ *and* $(\neg q_1, \neg c)$ *are two arguments over* Γ_1 *and* Γ_2, *respectively, and the argument* $(\neg q_1, \neg c)$ *attacks the argument* $((p_1, p_2, p_3), c)$.

We will prove in the "Complexity analysis" section that supposing the consistency of the Horn knowledge bases makes the attack operation tractable.

Definition 4 (argumentation system for a Web service). *This system is a pair* $\langle \mathcal{A}, \mathcal{AT} \rangle$, *where* $\mathcal{A}$ *is a set of arguments and* $\mathcal{AT} \subseteq \mathcal{A} \times \mathcal{A}$ *is the attack relationship. For a subset S of arguments in the argumentation system* $\langle \mathcal{A}, \mathcal{AT} \rangle$,

1. $a \in \mathcal{A}$ *is acceptable with respect to S if* $\forall b \in \mathcal{A}$ *such that b* $\mathcal{AT}$ *a and* $\exists c \in S$ *such that c* $\mathcal{AT}$ *b.*
2. *S is conflict free if no argument in S is attacked by any other argument in S.*

Dialogue games and community management

In the following, Ag_{WS} stands for agent of a Web service and $KB(Ag_{WS})$ stands for the knowledge base this agent uses to store details (precisely, beliefs) on this Web service. The agents' knowledge bases are conflict free (see definition 4).

Formal foundations. To persuade a Web service to be part of a community and to negotiate its participation in a given composite Web service, master and slave Web services use persuasion and negotiation techniques associated with their argumentation abilities. Hereafter, we specify a Horn logic-based persuasion and negotiation protocol. For flexibility requirements, this protocol is specified as a combination of a set of initiative and reactive dialogue games.[7]

Dialogue games are interaction games in which each agent makes a move by performing utterances according to a predefined set of rules.[8] The idea is to specify dialogue games as small rules expressed in terms of dialogue moves and conditions (see definition 5). The whole protocol is dynamically built by combining different dialogue games. This protocol represents a combination of Douglas Walton and Erik Krabbe's persuasion and negotiation dialogues.[9]

From a logical viewpoint, moves in a game are expressed as Horn-based communicative actions that agents perform by producing utterances and arguments. To make the protocol simple, we proposed a small set of communicative actions: Assert, Accept, Refuse, Challenge, Justify, Attack, and Defend. These actions are general in that they contain general proposals (ideas, offers, and so forth). For example, in the case of persuasion, an Assert could be an idea. In a negotiation, it could be an offer, in which case it would play the role of *Propose* or *Make Offer*. Similarly, an Attack could be an offer or a counteroffer.

Definition 5 (dialogue game). *Let* $Action_{Ag_{WS_1}}$ *and* $Action_{Ag_{WS_2}}$ *be two Horn-based communicative actions performed by* Ag_{WS_1} *and* Ag_{WS_2}, *respectively, and let Cond be a formula from the Horn logical language* £. *A dialogue game for an agent-based Web service is a set of logical rules indicating that if* Ag_{WS_1} *performs* $Action_{Ag_{WS_1}}$, *and if Cond is satisfied, then* Ag_{WS_2} *will afterwards perform* $Action_{Ag_{WS_2}}$. *We express these rules as*

$$Action_{AgWS_1} \xrightarrow{Cond} Action_{AgWS_2}$$

We express *Cond* in terms of the possibility of generating an argument from an agent's argumentation system. We identify two types of arguments: *private arguments*, which an agent doesn't reveal, and *public arguments*, which an agent exchanges during conversation. The following sets are introduced:

$$PrSupport(Ag_{WS}, p) = \{p' \mid p' \vdash p\}$$

$$PbSupport(Ag_{WS}, p) = \{Assert(Ag_{WS}, q) \mid q \vdash p\}$$

$PrSupport(Ag_{WS}, p)$ is the set of Ag_{WS}'s private arguments supporting the Horn proposition p. $PbSupport(Ag_{WS}, p)$ is the set of commitments Ag_{WS} creates to support p. $Assert(Ag_{WS}, q)$ is the Assert communicative action performed by Ag_{WS}, and q is the content. This set is closed under the support relation—that is,

$$p_2 \in PbSupport(Ag_{WS}, p_1) \wedge p_1 \in PbSupport(Ag_{WS}, p_0) \Rightarrow p_2 \in PbSupport(Ag_{WS}, p_0)$$

The formula $p \lhd Arg\_Sys(Ag_{WS})$ expressed in Horn language £ denotes that Ag_{WS}'s argumentation system, $Arg\_Sys(Ag_{WS})$, can generate a Horn propositional formula p. The formula $\sim (p \lhd Arg\_Sys(Ag_{WS}))$ indicates that $Arg\_Sys(Ag_{WS})$ can't generate p (~ represents the negation as failure). An agent's argumentation system can generate a Horn propositional formula p if the agent can find an argument supporting p.

> Dialogue games are interaction games in which each agent makes a move by performing utterances according to a predefined set of rules.

Specification of dialogue games. Dialogue games are specified as a set of logical rules whose conditions are expressed in terms of arguments. These rules indicate the possible actions that agent-based Web services can perform according to current situations. However, this specification doesn't describe which strategy an agent-based Web service should select. A strategy determines

- which arguments an agent should forward to persuade another peer,
- how to present these arguments (or which argument to present if many are possible), and
- how to combine dialogue games to best persuade the addressee.

For example, in an attack scenario, if a Web service has several counterarguments, the strategy identifies which one to choose on the basis of increasing the chances for acceptance. This separation of dialogue games and strategies is important. Dialogue games are public and should be shared by all Web services, but strategies are private. Agent compliance with a dialogue game protocol is verifiable because conditions are specified in terms of generating veri-

fiable arguments (we can check to see if an agent has an argument for or against a proposition).

Three types of dialogue games support agents performing communicative actions such as Assert, Challenge, and Defend:

- The *entry game* initiates conversation.
- The *chaining game* continues a conversation by combining several dialogue games—defense, attack, challenge, and justification games.
- The *termination game* ends a conversation when the exit conditions are satisfied.

In the entry game, a master Web service can engage in conversation a new Web service registered in a given UDDI registry. If the new Web service accepts, then the master Web service will start another conversation to persuade it to join the community. In the same community, a slave Web service can invite other slave Web services to negotiate their participation in a composite Web service. We specify this entry game as follows:

$$Assert(Ag_{WS_1}, p) \xrightarrow{C_1} Accept(Ag_{WS_2}, p)$$

$$Assert(Ag_{WS_1}, p) \xrightarrow{C_2} Refuse(Ag_{WS_2}, p)$$

where

$$C_1 = (p \lhd Arg\_Sys(Ag_{WS_2})) \vee \sim(\neg p \lhd Arg\_Sys(Ag_{WS_2}))$$

$$C_2 = \neg p \lhd Arg\_Sys(Ag_{WS_2})$$

Proposition p is expressed £ using a shared ontology. This proposition is an invitation to start a conversation. If the invited Web service has an argument in favor of p or doesn't have any argument against p, it accepts the invitation; otherwise, it refuses. For example, if a new Web service isn't interested in joining a community owing to previous "bad" experiences, it sends a refusal to the master Web service. If a Web service doesn't have any information about the community or believes that the community's configuration is efficient, it accepts the invitation.

An important dialogue game in persuasion and negotiation interactions is the defense game. A Web service adopts this game to defend the propositions (in a persuasion) or offers (in a negotiation) it makes. For example, a master Web service defends its invitation to a new Web service with various arguments, such as the participation rate of other slave Web services in composite Web services, the community's efficient configuration, or the rationale for having this Web service. We specify the defense game as follows:

$$Defend(Ag_{WS_1}, (H,h)) \xrightarrow{C_1} Accept(Ag_{WS_2}, h)$$

$$Defend(Ag_{WS_1}, (H,h)) \xrightarrow{C_2} Challenge(Ag_{WS_2}, H)$$

$$Defend(Ag_{WS_1}, (H,h)) \xrightarrow{C_3} Attack(Ag_{WS_2}, (H',h'))$$

where

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

$$C_2 = \sim(H \lhd Arg\_Sys(Ag_{WS_2})) \wedge \sim(\neg H \lhd Arg\_Sys(Ag_{WS_2}))$$

$$C_3 = (H' \lhd Arg\_Sys(Ag_{WS_2})) \wedge (H',h')\mathcal{AT}(H,h)$$

We define generation of a set of Horn formulas H from Ag_{WS_2} as follows:

$$H \lhd Arg\_Sys(Ag_{WS_2}) \triangleq \forall h_i \in H h_i \lhd Arg\_Sys(Ag_{WS_2})$$

$$\neg H \lhd Arg\_Sys(Ag_{WS_2}) \triangleq \exists h_i \in H \neg h_i \lhd Arg\_Sys(Ag_{WS_2})$$

By definition, $Defend(Ag_{WS_1}, (H,h))$ means that Ag_{WS_1} asserts argument (H, h) to defend h, which could be an opinion or an offer. $Attack(Ag_{WS_2}, (H',h'))$ means that Ag_{WS_2} asserts argument (H', h') to attack argument (H, h). Ag_{WS_2} accepts Ag_{WS_1}'s argument if it can generate this argument out of its knowledge base. If Ag_{WS_2} can't generate a new argument for or against this argument, it challenges the argument by requesting further explanations (the challenge game). Finally, Ag_{WS_2} attacks the argument if it can generate an attacker argument (the attack game).

We specify the challenge game as follows:

$$Challenge(Ag_{WS_1}, H) \xrightarrow{C_1} Justify(Ag_{WS_2}, (H',H))$$

where

$$C_1 = PrSupport(Ag_{WS_2}, H)$$

Condition C_1 should always be true, because a Web service must always be able to justify its propositions and assertions.

We specify the justification game as follows:

$$Justify(Ag_{WS_1}, (H,h)) \xrightarrow{C_1} Accept(Ag_{WS_2}, h)$$

$$Justify(Ag_{WS_1}, (H,h)) \xrightarrow{C_2} Challenge(Ag_{WS_2}, H)$$

$$Justify(Ag_{WS_1}, (H,h)) \xrightarrow{C_3} Attack(Ag_{WS_2}, (H',h'))$$

Similar to the defense game, in the justification game, Ag_{WS_2} can accept, challenge, or attack Ag_{WS_1}'s justification. C_1, C_2, and C_3 are identical to the conditions in the defense game.

Finally, we specify the attack game as follows:

$$Attack(Ag_{WS_1}, (H,h)) \xrightarrow{C_1} Accept(Ag_{WS_2}, h)$$

$$Attack(Ag_{WS_1}, (H,h)) \xrightarrow{C_2} Challenge(Ag_{WS_2}, H)$$

$$Attack(Ag_{WS_1}, (H,h)) \xrightarrow{C_3} Attack(Ag_{WS_2}, (H',h'))$$

$$Attack(Ag_{WS_1}, (H,h)) \xrightarrow{C_4} Refuse(Ag_{WS_2}, h)$$

where

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

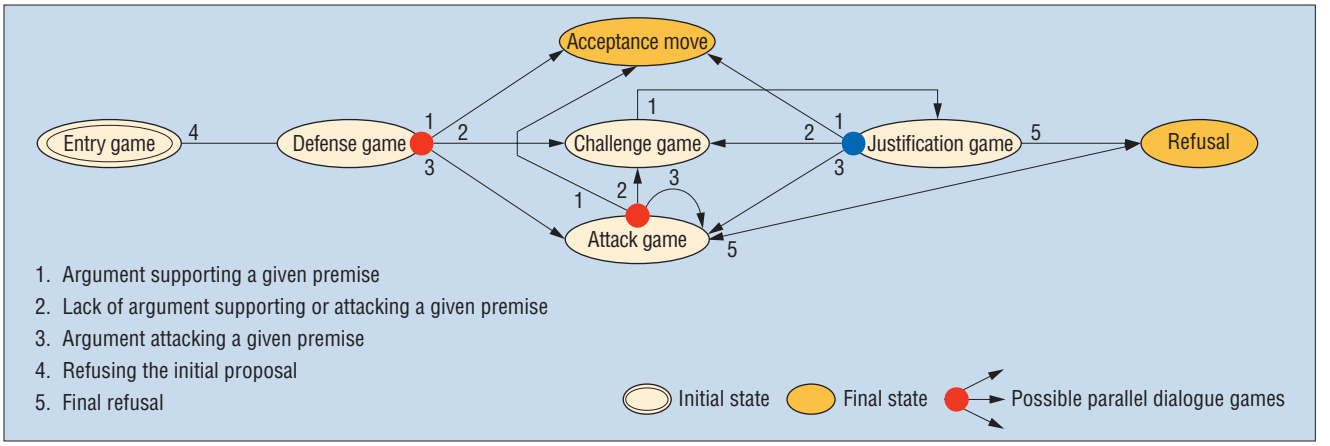

Figure 2. The Persuasion/Negotiation for Agent-Based Web Services protocol dynamics as a finite-state machine.

$$C_2 = \sim(H \lhd Arg\_Sys(Ag_{WS_2})) \wedge \sim(\neg H \lhd Arg\_Sys(Ag_{WS_2}))$$

$$C_3 = (H' \lhd Arg\_Sys(Ag_{WS_2})) \wedge (H', h')\mathcal{AT}(H, h)$$

$$C_4 = \neg h \in KB(Ag_{WS_2})$$

An agent-based Web service Ag_{WS_2} accepts an attacker's argument if it can generate a support using its argumentation system. If it can't generate or negate this support, the agent challenges it. If it can generate a counterattack argument, then it plays the attack game. Otherwise, it refuses the attacker's argument. It can play this *refuse move* if the negation of the attacker's argument conclusion is in Ag_{WS_2}'s knowledge base. In that case Ag_{WS_2} can't play the attack game; it doesn't have a counterargument—only knowledge about the negation of the argument conclusion. This possibility of a refusal move makes defense and attack different games. Agents play defense games only after they agree to engage in persuasion or negotiation (by replying to the entry game). Consequently, the invited Web service can attack the inviter's conclusion.

We now discuss how to combine the different dialogue games. During a conversation, an agent can't play the same move (or use an argument) more than once—reiterations are prohibited. Also, dialogue games can be played sequentially or in parallel. For example, a Web service can accept one part of the argument presented by another Web service and challenge another part in parallel. The conversation terminates by either a final acceptance or refusal. There's a final acceptance when a Web service accepts the initial proposition (for example, to join the community) or when an agreement is reached.

The Persuasion/Negotiation for Agent-Based Web Services (PNAWS) protocol that combines these games is specified using the Backus–Naur Form grammar, although using other formalisms (such as the Agent-based Unified Modeling Language) wouldn't affect our proposals. However, because the protocol is a combination of logical rules, using a language-driven formalism such as BNF is more expressive, particularly for specifying parallel games and nondeterministic choices. We first introduce possible parallel dialogue games.

DEFINITION 6 (possible parallel dialogue games). *Let $G_{1,2,3}$ be three dialogue games, and ϵ, "|", ";", and "//" be the empty dialogue game, the choice symbol, the sequence symbol, and the parallelization symbol, respectively. G_1// G_2 means that an agent can play two games in parallel. The specification of possible dialogue games is*

$$//(G_1; G_2; G_3) \triangleq ((G_1 //_{\geq 1} G_2) //_{opt} G_3) \mid ((G_1 //_{opt} G_2) //_{\geq 1} G_3)$$

where

$$G_1 //_{\geq 1} G_2 \triangleq G_1 \mid G_2 \mid (G_1 // G_2)$$

$$G_1 //_{opt} G_2 \triangleq \epsilon \mid (G_1 //_{\geq 1} G_2)$$

Let's assume that the entry game is successful (accepted); we now define the PNAWS protocol as follows (where WSDG stands for Web Service Defense Game):

PNAWS = entry game; defense game; WSDG
WSDG = // (acceptance move; Ch; Att)
Ch = challenge game; justification game; (WSDG | Refusal)
Att = attack game; (WSDG | Refusal)

A finite-state machine could describe the PNAWS protocol dynamics (see figure 2). The acceptance move, challenge game, and attack game are possible parallel dialogue games as specified in definition 6. So, agents could play all three in parallel, play just one game, or play two in parallel without playing the third. The state machine describes how an agent-based Web service could use the protocol. The state machine models the behavior of one agent when that agent receives the communicative actions from the addressee. Each agent will, individually, use the same protocol, because that protocol is described independently from the agents and their roles.

Proposition 2. *For any set of dialogue games, the PNAWS protocol always terminates.*

Proof. *The PNAWS protocol is defined recursively because Ch and Att*

are defined in terms of WSDG. Because the same move is prohibited during a conversation, and the content of communicative acts has a finite size, challenge and attack moves are finite. In addition, because the agent-based Web service's knowledge bases are finite, justification moves are finite. Consequently, the protocol terminates by executing either a final refusal or a final acceptance.

Complexity analysis

Here we prove that the suggested argumentation-based reasoning for agent-based Web services is tractable.

Proposition 3. *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,*[10] *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.*

Deciding whether an argument for a Web service supports a conclusion over a definite Horn knowledge base is polynomial. This result follows immediately from the following proposition:

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

Proof. *If* (H, h) *is an argument where H is a set of definite Horn formulas under the form c or* $p_1 \wedge p_2 \wedge \ldots \wedge p_n \rightarrow c$*, 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* $H' : \Gamma \supseteq H' \supseteq H$*. Because* $H \vdash h$*, it follows that* $H' \vdash h$*, whence the proposition.*

Theorem 1. *Given a definite Horn knowledge base* Γ *and a formula h. Deciding whether there's an argument* (H, h) *is polynomial.*

Proof. *From Proposition 4, it follows that there is an argument supporting h if and only if* $(\Gamma, h) \in \mathcal{A}$*. So, by proposition 3, the theorem follows.*

The following corollary is a direct consequence of theorem 1.

Corollary 1. *Given a consistent Horn knowledge base* Γ *and a formula h, deciding whether there's an argument* (H, h) *is polynomial.*

Proof. *Proposition 4 holds if* Γ *is consistent. Then, by proposition 3, the result follows.*

This positive result is important for implementing Web services and communities of Web services using argumentation-based agents specified in Horn logic. In fact, maintaining the consistency of Horn knowledge bases is tractable. If Γ is a general Horn theory, the following theorem holds.

Theorem 2. *Given a Horn knowledge base* Γ *and a formula h, deciding whether there's an argument* (H, h) *is NP-complete.*

Proof. *Membership of this problem in NP is straightforward because by proposition 3, checking that a guessed subset* $H \subseteq \Gamma$ *is consistent and* $H \vdash h$ *could be done in polynomial time.*

To prove the problem's hardness, we use a transformation from the well-known SAT (Boolean satisfiability) problem. Let $C = \{C_1, C_2, \ldots, C_n\}$ *be a set of propositional clauses on* $X = \{x_1, x_2, \ldots, x_n\}$ *and let* $X' = \{x'_1, x'_2, \ldots, x'_n\}$, $Y = \{y_1, y_2, \ldots, y_m\}$*, be sets of new variables. So, there's an argument* (H, h) *over* Γ*, where h is a formula over Y and*

$$\Gamma = X \cup X' \cup \{\neg x_i \vee \neg x'_i : 1 \le i \le n\} \cup \bigcup_{i=1}^{m} (\{x_j \rightarrow y_i : x_j \in C_i\} \cup \{x'_j \rightarrow y_i : \neg x_j \in C_i\})$$

if and only if C is satisfiable. Because Γ *is constructible in polynomial time, we're done.*

Even when using general Horn logic results in intractable reasoning, this result is still better than the one obtained using propositional logic, in which the complexity is $\sum_2^p$*−complete*.

Now, we discuss the attack operation's complexity.

Proposition 5. *Given a Horn knowledge base* Γ*, a subset H of* Γ*, a formula* $x \in H$*, and a formula h. Deciding whether x is relevant to* (H, h) *is polynomial.*

Proof. *x is relevant to* (H, h) *if and only if* $H - \{x\} \not\vdash h$*. This can be done in polynomial time.*

Proposition 6. *Let* Γ_1 *and* Γ_2 *be two Horn knowledge bases and* (H, h) *and* (H', h') *be two arguments over* Γ_1 *and* Γ_2*, respectively. Deciding whether* (H', h') $\mathcal{AT}(H, h)$ *is polynomial.*

Proof. *The first condition of definition 3* $(H' \vdash \neg h)$ *is polynomial. The first part of the second condition* $(\exists x \in H : H' \vdash \neg x)$ *is decidable in* $O(|H| \times |H'| \times |\neg x|)$*, and the second part* (x *is relevant to* (H, h)) *is polynomial by proposition 5, whence the proposition.*

Theorem 3. *Let* Γ_1 *and* Γ_2 *be two consistent Horn knowledge bases and* (H, h) *an argument over* Γ_1*. Deciding whether there's an attacker of* (H, h) *over* Γ_2 *is polynomial.*

Proof. *From definition 3, there's an attacker of an argument* (H, h) *if and only if another argument exists that negates its conclusion or one of the relevant premises. From corollary 1, deciding whether there is an argument* $(H', \neg h)$ *is polynomial. Also, from the same corollary and proposition 5, deciding for each* $x \in H$ *relevant to* (H, h) *whether there's an argument* $(H', \neg x)$ *is polynomial. So, the whole problem is decidable in polynomial time.*

PNAWS against McBurney and colleagues' desiderata

A relevant set of desiderata that permit assessing argumentation-based dialogue games appears elsewhere.[11] The following illustrates how PNAWS meets the desiderata:

- *Stated dialogue purpose:* PNAWS is explicitly used for persuasion and negotiation.
- *Diversity of individual purposes:* PNAWS lets agents achieve their own purposes in terms of persuading and negotiating with peers.
- *Inclusiveness:* There's no elimination of agents.
- *Transparency:* All agents share PNAWS's rules and structure.

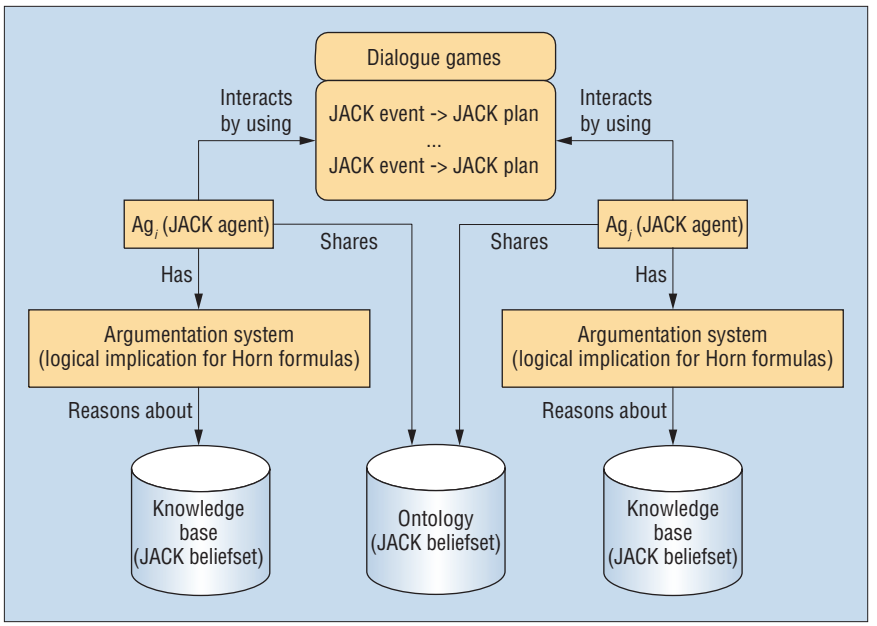

Figure 3. The system architecture for our proposed dialogue game protocol.

- *Fairness:* Slave Web services are treated equally, and the asymmetry between them and master Web services is explicit.
- *Clarity of argumentation theory:* The argumentation theory is explicit and clear *in* PNAWS's specification.
- *Separation of syntax and semantics:* The PNAWS's specification language separates syntax from semantics. The syntax is expressed in terms of actions and conditions, and the semantics is defined in terms of argumentation theory.
- *Rule consistency:* PNAWS specifies all rules and the conditions, expressed in terms of arguments, guarantee consistency. Furthermore, there are no infinite cycles or repeated locutions. PNAWS termination is formally proved.
- *Encouragement of resolution:* The rules don't eliminate normal termination.
- *Discouragement of disruption:* PNAWS precludes disruptive behavior by prohibiting the performance of the same communicative acts repeatedly and the use of the same arguments.
- *Enablement of self-transformation:* PNAWS enables self-transformation by allowing acceptance moves, so agents can accept addressees' arguments and update their knowledge bases.
- *System simplicity:* PNAWS includes only seven locutions, which keeps PNAWS simple.
- *Computational simplicity:* It's proved that the reasoning is tractable and computationally simple (polynomial complexity).

PNAWS is fully specified and some criteria are formally proved. In particular, its computational simplicity is significant, because this isn't the case in many other dialogue game protocols, some of which are even intractable.

Argumentation automation

We implemented our dialogue game protocol to provide a practical proof that argumentation-based reasoning using propositional Horn logic for Web services is tractable. We used XML and Java development kit version 1.4 for operation processing. We used the JACK Intelligent Agents framework, equipped with logical inference, for Horn formulas to implement agents, their knowledge bases, and argumentation-driven reasoning. We also used the JACK framework to specify dialogue games. Figure 3 shows the system architecture.

The prototype shows how agent-based Web services operate on top of their knowledge bases, update their knowledge bases during and after conversations, and generate arguments for and against propositions. Moreover, the prototype simulates how agents trigger persuasion scenarios that lead Web services to, for example, join communities or participate in compositions.

In the prototype, each agent-based Web service has, as part of its internal structure, a knowledge base of beliefs and an argumentation system it uses to reason about these beliefs by selecting and combining dialogue games. Agents share the same ontology for defining literals.

After several simulations, we noted that generating arguments and converging toward agreements in persuasions and negotiations are faster when the agents' knowledge bases are small. Also, reaching agreements between a master Web service and a new Web service about joining a community is faster when the number of candidate communities is small.

Dialogue games are declaratively specified outside agents' structure and stored in a shareable store. Agents refer to dialogue games' specifications when they need to determine their next moves according to the argumentation system.

We implemented agent knowledge bases as a set of tables. Additionally, we implemented beliefs in these knowledge bases as data structures called *beliefsets* and represent them using Horn logic and a tuple-based relational model. JACK automatically maintains the logical consistency of the beliefs in a beliefset. So, if an agent believes that p is true and, after a persuasion phase with another agent accepts $\neg p$, the agent can update its beliefset by removing p and all arguments supporting p and adding $\neg p$ to the knowledge base.

On top of beliefs, the knowledge base contains arguments that have the form **([Support],Conclusion)**, where **Support** is a set of Horn clauses and **Conclusion** is a Horn formula. The meaning of literals according to the ontology is recorded in a beliefset implemented in JACK as a **table_ontology** that contains two fields: **Literal** and **Meaning**. All agents share access to the **table_ontology**.

Agents communicate through messages that are events, which extend JACK's **MessageEvent** class. An agent can send a **MessageEvent** using a **Send(Destination, Message)** primitive. We implemented each dialogue game as a set of events (or **(MessageEvents)**) and plans. A plan describes a sequence of actions that an agent can perform when an event occurs. Whenever an event is posted, the addressee uses its argumentation system to find a plan that would handle this event. Games aren't implemented in the agents' structure but as independent event classes and plan classes. So, each interacting agent-based Web service can instantiate these classes.

Execution is as follows. Initially, Ag_{WS_i} uses its argumentation system to generate an argument supporting its initial proposition—let's say to invite a new Web service to join the community. It starts a dialogue game according to the shared dialogue game description by generating an event that corresponds to a move. It sends this event to the interlocutor, Ag_{WS_j}. Ag_{WS_j} then uses its argumentation system to determine its move (in this game that Ag_{WS_i} started) according to the shared dialogue game description. Ag_{WS_j} executes its plan corresponding to this move, generates another event, and sends the event to Ag_{WS_i}. This starts a new dialogue game—perhaps Ag_{WS_j} refuses some of the inviter's Ag_{WS_i}'s joining conditions or rewards. Ag_{WS_i} then determines the next dialogue game in accordance with the dialogue game description, and so on until the dialogue terminates. Consequently, both agents can communicate using the same protocol, because they can instantiate the same classes representing all events and plans. For example, `Event_Attack` and `Plan_ev_Attack` implement the Attack game.

Combining argumentation theory and Web services opens up multiple research opportunities. For example, the dynamic nature of a community calls for adaptable security mechanisms that could be based on trust networks and role-based trust management. Another opportunity is to examine alliances of Web services as a means for structuring communities and forming collaborative partnerships among Web services in communities. ■

References

1. Z. Maamar et al., "Web Services Communities—Concepts & Operations," *Proc. 3rd Int'l Conf. Web Information Systems and Technologies* (WEBIST 07), 2007, pp. 323–327.
2. I. Rahwan, "Guest Editorial: Argumentation in Multi-Agent Systems," *J. Autonomous Agents and Multiagent Systems*, vol. 11, no. 2, 2006, pp. 115–125.
3. B. Benatallah, Q.Z. Sheng, and M. Dumas, "The Self-Serv Environment for Web Services Composition," *IEEE Internet Computing*, vol. 7, no. 1, 2003, pp. 40–48.
4. C.I. Chesñevar, A. Maguitman, and R. Loui, "Logical Models of Argument," *ACM Computing Surveys*, vol. 32, no. 4, 2000, pp. 337–383.
5. H. Prakken and G.A.W. Vreeswijk, "Logics for Defeasible Argumentation," *Handbook of Philosophical Logic*, 2nd ed., Kluwer Academic, D.M. Gabbay and F. Guenthner, eds., vol. 4, 2002, pp. 219–318.
6. S. Parsons, M. Wooldridge, and L. Amgoud, "Properties and Complexity of Some Formal Inter-agent Dialogues," *J. Logic and Computation*, vol. 3, no. 13, 2003, pp. 347–376.
7. J. Bentahar, B. Moulin, and B. Chaib-draa, "Specifying and Implementing a Persuasion Dialogue Game using Commitments and Arguments," *Argumentation in Multi-Agent Systems*, vol. 3366, Springer, 2005, pp. 130–148.
8. P. McBurney and S. Parsons, "Games That Agents Play: A Formal Framework for Dialogues between Autonomous Agents," *J. Logic, Language, and Information*, vol. 11, no. 3, 2002, pp. 315–334.
9. D. Walton and E. Krabbe, *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*, State University of New York Press, 1995.
10. W. Dowling and J.H. Gallier, "Linear-time Algorithms for Testing the Satisfiability of Propositional Horn Theories," *J. Logic Programming*, vol. 1, no. 3, 1984, pp. 267–284.
11. P. McBurney, S. Parsons, and M. Wooldridge, "Desiderata for Agent Argumentation Protocols," *Proc. 1st Int'l Conf. Autonomous Agents and Multi-Agent Systems*, 2002, pp. 402–409.

The Authors

Jamal Bentahar is an assistant professor of computer science and software engineering at the Concordia Institute for Information Systems Engineering at Concordia University, Montreal. His research interests include multiagent systems, argumentation theory, logic and formal methods, Web services, and grid computing. He received his PhD in computer science from Laval University. He's a member of the IEEE and ACM. Contact him at Concordia Univ., Concordia Inst. for Information Systems Eng., 1455 de Maisonneuve Blvd. West, EV007.640, Montreal, QC, H3G 1M8, Canada; bentahar@ciise.concordia.ca.

Zakaria Maamar is an associate professor at the college of information technology at Zayed University. His research interests include Web services, software agents, and context-aware computing. He received his PhD in computer science from Laval University. Contact him at Zayed Univ., College of Information Technology, Dubai Campus, PO Box 19282, Dubai, United Arab Emirates; zakaria.maamar@zu.ac.ae.

Djamal Benslimane is a full professor of computer science at Claude Bernard Lyon 1 University. His research interests include interoperability, Web services, and ontologies. He received his PhD in computer science from Blaise Pascal University. He's a member of the Laboratoire d'InfoRmatique en Images et Systèmes d'information—Centre National De la Recherche Scientifique (LIRIS-CNRS). Contact him at the Univ. of Claude Bernard Lyon 1, LIRIS Laboratory, IUT A Informatique, 18, Bld Niels Bohr 69622, VIlleurbanne Cedex, France; djamal.benslimane@liris.cnrs.fr.

Philippe Thiran is an associate professor of management information systems in the University of Namur's Department of Business Administration. His research interests include Web services, databases, and distributed information systems. He received his PhD in computer science from the University of Namur. He's a member of the PReCISE Research Center. Contact him at the Dept. of Business Administration, FUNDP-Univ. of Namur, 8, rempart de la Vierge, B-5000 Namur, Belgium; pthiran@fundp.ac.be.