Enhancing Engineering Methodology for Communities of Web Services

M. El-Menshawy
Depart. of Electrical and Computer Engineering
Concordia University, Montreal, Canada
m_elme@encs.concordia.ca

J. Bentahar, R. Dssouli
Concordia Institute for Inf. Sys. Engineering
Concordia University, Montreal, Canada
{bentahar, dssouli}@ciise.concordia.ca

***Abstract*—Communities of web services have been proposed to gather web services having the same functionalities but possibly different nonfunctional properties. Current approaches into communities of web services focus on developing, managing and designing communities of web services through a suitable architecture, but can benefit from a stronger treatment of flexible interactions. These approaches ignore the collaboration and business-level contracts between various web services and the ability to formally delegate service to another web service within the same community. This paper presents a significant step towards enhancing communities of web services using an agent-based approach that synthesizes mentalistic states (e.g. goals, tasks), social commitments and argumentative dialogues for modeling and establishing communities of web services. This paper has three contributions: first, we extend the community structure with alliances structure to allow collaboration between various web services; second, we propose a new engineering methodology based on concepts of Tropos methodology for managing communities of web services with alliances structure; and third, we specify internal-organizational business interactions within web services in terms of commitments augmented with argumentative dialogues to reason about the validity of these commitments. We evaluate our methodology using a large existing case study of auto insurance claim processing.**

I. Introduction

The notion of community has been proposed to gather web services having the same functionalities independently of their origins and the way they carry out these functionalities [3], [4], [11]. In recent years, the capability of argumentation and dialogue games has been used in managing and reconciling conflicts of interests that may arise within a community of web services. In this context, the argumentation theory allows agent-based web services to interact rationally, argue about the reasons that support or disavow their conclusions, persuade a new web service to join a community, negotiate with other peers to reach a deal and assail each other through an attack-binary relation as well as specify the interaction mechanisms within communities [3], [4].

In fact, web services of business scenarios begin with a user's need and end with a user's need fulfillment. The structure of community facilitates and speeds up the process of web services discovery in open settings and helps in selecting the best ones for composite business scenarios [12], [14] when users' requests cannot be satisfied by a single available web service but need collaboration among available web services to handle them [11].

Recently, a certain number of significant proposals have been introduced [1], [3], [4], [10]–[13] to address the modeling and management issues of communities of web services (CWSs) in order to allow them to interact more flexibly for the growing needs of business processes. However, the main objective of modeling and managing CWSs and the collaboration between web services in an efficient way along with business relationships has not been reached yet. In particular, we have three broad elements which should be addressed: designing, engineering, and managing communities.

- The architecture based on traditional software engineering methodologies proposed in [3], [4], [11] lacks business-level contracts that represent underlying interactions between web services. This architecture depends on message occurrence and ordering irrespective of the message meanings, thereby a message-based approach hides many details of the internal organization of the architecture that affect designing business processes.
- Existing methodology [4], [11] for modeling, operationalizing, and evolving CWSs ignores formalizing the delegation process that is used to delegate incomplete services within community, in the case of replacing misbehaving web services and keeping the system reliable. Furthermore, this methodology does not consider the collaboration among members of the community and the mentalistic states of agent-based web services. In general, the approaches discussed in the literature consider only two-party operations, while real-life scenarios are typically multiparty operations that need a mediator agent-based web service to complete users' requests.
- The process of handling failures and exceptions is very hard to implement within the current structure of the community. Moreover, the number of interactions between members of the community needs to be reduced to enhance the response time of participating web services.

This paper aims to enhance the community structure by setting up alliances structure among web services to overcome the aforementioned shortcomings in [3], [4], [11]. Thereby the resulting community structure becomes more realistic with the real-life business scenarios in distributed systems. By so doing, we propose a new engineering methodology based on Tropos methodology [17], which we enhance with concepts

of communities, commitments and argumentative dialogues for modeling CWSs extended with alliances structure. In fact, this work is an extension of our previous research in which we have modeled and specified CWSs based on argumentation capabilities. This model enables web services through associated agents to argue, persuade and negotiate with their peers using a dialectical process to satisfy their goals in an efficient way [3], [4]. Specifically, here we reconfigure the design of community with alliances structure from the perspective of the collaboration between agent-based web services that participate in composition business scenarios and benefit from agent reasoning capabilities about nonfunctional properties.

The contributions of this paper are manifold: (i) alliances structure within CWSs enhances response time of web services that handle users' requests and reduces the number of interconnections between members of community; (ii) a new agent-based engineering methodology synthesizing mentalistic states, social commitments and argumentative dialogues; and (iii) a composition mechanism allowing agent-based web services to collaborate with each other in the form of delegation operations. Additionally, we present a case study of auto insurance industry scenario to evaluate our methodology.

The remainder of this paper is organized as follows. Section II introduces the notion of alliance structure and key concepts for our methodology. Section III presents our engineering methodology for developing communities of web services. Our case study is presented in Section IV to evaluate different steps of our methodology. The paper ends in Section V with the relevant literature discussions and future work directions.

II. Alliance Structure and Concepts

This section defines alliances structure within communities of web services and key concepts to be used clearly in our methodology.

A. Alliance Structure

In [3], [4], [11] the notion of communities of agent-based web services has been introduced. Here we would prescribe how to extend it via introducing alliances structure. In essence, the service providers over forced to improve their strategies and redistribute business functionalities to be able to compete with others should think about building alliances. An alliance structure based on Quality of Service (QoS) is the concept that reconfigures community structure to achieve competitive pressures between service providers and collaboration between agent-based web services. In [3], [11] an alliance structure as a subset of community or a micro-community based on mutual agreements between providers of web services as part of their partnership strategies has been superficially introduced.

We develop this view by considering nonfunctional proprieties (e.g., QoS, reputation, response time) associated with each agent-based web service as a vital principle to cluster two or more web services into different alliances as the second level of community, since the first level is occupied by master web service (see Fig.1). Whereas the third level of community contains web services within alliances that underpin the same or part of the community's functionality. These web services need to collaborate with other peers to achieve the whole or global community's functionality.

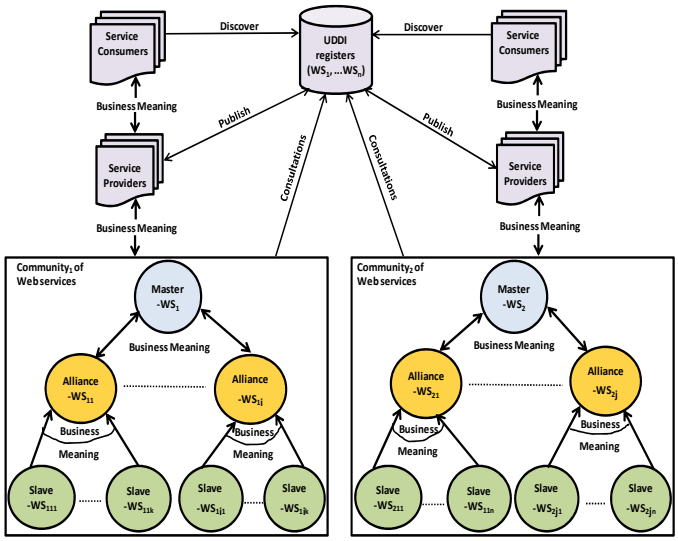

Fig. 1. An architecture of communities extended with alliances structure.

For example, in the purchase scenario of goods, the functionality of a community is purchasing goods. This community combines web services having complementary functionality within an alliance, such as *placing an order*, *paying*, and *shipping* with a high reputation and response time. Notice that, these agent-based web services have logically distinctive functionalities and involve distinctive roles from the community's functionality but they can collaborate with each other to achieve a goal by combining or composing their functionalities (i.e., purchasing goods) and to satisfy the community's functionality.

As a result the new internal organization of the community with alliances structure make it easier to detect failures and errors as we search in micro-communities and reduce the number of interactions between members of the community since the master web service has one connection with each alliance structure instead of a direct connection with each web service (i.e., this minimize the overhead). Moreover, an alliance inherits a dynamic property from its community where new members can admit to or exclude from the alliance, and alliances themselves can either discard or merge at any-time.

B. Key Concepts

The key concepts of our methodology are inspired by Tropos methodology [17] with the extra flexibility resulting from considering new concepts such as community, commitment and argumentative dialogue. The purpose is to enhance Tropos capabilities in order to deal with the intrinsic complexity of business processes.

1) ***Community****:* a collection of web services with similar or part of total functionality organized into three broad levels without explicitly referring to concrete web services that will implement this functionality within alliances structure at runtime (see Fig.1).

*2) **Agent**:* a computational representation of web service within community structure with metaphors that make it appropriate for developing, designing and implementing distributed business systems. Each agent has strategic goals and capabilities to execute tasks. It engages with others in business relationships to get other agents performing the delegated tasks on its behalf and to reason about the validity of their tasks.

*3) **Role**:* an abstract entity over agent-based web services and can be specified by their related sets of commitments augmented with argumentation systems within a community. Multiple agents could play one role and different roles could be played by a single agent in a community.

*4) **Goal**:* a state of the world that an agent would like to reach or bring about. In other words, a goal is a final or an acceptance state. Tropos methodology defines two types of goals: *hard goals* are functional requirements and often have a measurable satisfaction condition that can be satisfied. The latter one, called *soft goals*, no precise criteria for its satisfaction can be found, these goals model nonfunctional requirements of the community. The AND/OR decomposition is used to decompose a root goal into subgoals.

*5) **Task**:* an abstract method by which a goal can be achieved. Parallel with the concept of goal, there are two types of tasks to achieve the companion goals. Here we also use AND/OR decomposition to decompose a root task into subtasks compatible with subgoals. The AND requires all subtasks; OR requires one subtask.

*6) **Dependency**:* used to identify the dependent relationship between two roles where one role (the depender) depends upon the other (the dependee) in order to achieve a goal or execute a task. This relation is written as $the\ depender$ depends upon $the\ dependee$.

*7) **Commitment**:* a commitment $C(id, dbtr, cdtr, C_x, \psi, \phi)$ means that the debtor $dbtr$ is responsible to the creditor $cdtr$ within community context C_x for satisfying the content ϕ if the condition ψ holds. The commitment has the form of contractual style where id is the unique identifier for commitment, $dbtr$ is the debtor role, $cdtr$ is the creditor role, the context C_x may be an institution, a company, organization, marketplace (e.g., eBay) in which ongoing interactions occur, and ϕ and ψ are formulas in a given formal language.

Commitments capture state of the dependencies relation between roles and allowing a variety of possible manipulation based on a set of operations. For instance, the debtor is able to *create*, *fulfill*, *violate* commitments, *withdraw* from commitments and *delegate* commitments to another agent. Whilst the creditor has the right to *release* the debtor from commitments and *assign* commitments to another agent [20].

*8) **Argumentative dialogue**:* a dialectical process for the exchange of various arguments for and against some conclusions [3], [4]. Indeed, argumentation provides agent-based web services with an effective means to reconcile conflicts, seek information, persuade and negotiate with other peers within the same alliance structure. It relies on actions on commitments to generate a suitable set of arguments during dialogues to achieve mutually acceptable agreements between agent-based web services.

III. Engineering Methodology for web services Communities

Having captured the core concepts of our methodology, here we introduce the proposed methodology that intended to support all phases of developing CWSs based on the notion of Tropos methodology. The latter one has been developed in [17] as an agent-oriented software methodology in terms of goal, task and dependency. Tropos has been enhanced with commitments to capture business interactions among partners with high-level business meaning [15]. In the same direction, this paper improves the last version of Tropos introduced in [15] with arguments and argumentative dialogues to increase its capabilities via enabling agent-based web services to argue, seek information and negotiate with other peers about the compliance of their commitments, thereby increasing its practicality in distributed business systems. Table (I) summarizes the steps in our proposed methodology. The subsections below describe a step-by-step way the progress of our methodology.

A. Requirements Analysis

This phase enhances the early and late requirements in the phases of Tropos methodology with a community concept.

*1) **Step 1: Identify Community**:* the engineer initially concerns with understanding the organizational context of community that gathers agent-based web services. This step includes substeps to be completed.

1.1) Define the functionality (e.g., *hotels booking, weather forecasting, etc.*) of community by binding to a specific ontology [13] (e.g., Web Ontology Language (WOL)). This binding is important since providers of web services use different terminologies to describe the functional and nonfunctional requirements of their respective web services.

1.2) Identify agents and roles in a community using terms like $master$, $alliance$ and $slave$ web service. The master web service plays the main role in a community and refers to a special web service that leads the community as well as it takes over multiple responsibilities (e.g., checking the credentials of alliance web services before they are established in the community). The other of web services in the community are slave web services.

1.3) Identify alliance web services by clustering two or more slave web services having the same nonfunctional properties satisfy the users' requests and having different functionality. For example, the master web service gathers slave web services that have a high QoS in the first group under the management of *alliance web service1* and the other slave web services that have a medium QoS in the second group under the management of *alliance web service2*, etc. (see Fig.1).

1.4) Specifying dismantle community: the master web service is only responsible for dismantling community when all alliance structures present low precision to users' requests (i.e., irrelevant results) and low recall (i.e., missing relevant information). This happens when the number of slave web services within alliances structure is not enough to satisfy users' requests as well.

*2) **Step 2: Determine Goals and Dependencies**:* this step iteratively determines the goal dependencies between the roles. First, it searches UDDI registries to find posted services based on the similarity that exists between users' requests and these registered services. Thus, the composite goal of users' requests is identified at the master web service as a hard goal. Second, the master web service uses nonfunctional properties to refine its populated alliance web services into one that satisfies this composite goal (say G). Third, using means-end analysis to decompose this composite goal into subgoals (say $G = \{g_1, g_2, \ldots\}$), thereafter, alliance web service introduces roles that adopt these subgoals. This iterative analysis continues decomposing these subgoals until no new goal dependencies arise.

Description	Input	Output
1.1) Identify community function	A specific ontology and composition scenario.	All agent-based web services having the same or part of community's functionality.
1.2) Identify agents and roles in community	Agent-based web services and composition scenario.	Master web service and slave web services.
1.3) Identify alliance web services	Slave web services, composition scenario and nonfunctional properties.	Alliance web services and their slave web services.
1.4) Specify dismantling community	Request from alliance web service to its master web service.	Terminating community.
2) Identify goals and goal dependencies	Alliance web services, slave web services, composition scenario and goals are introduced in architecture.	Goals and goal dependencies of each dependee role (in step 1.2).
3) Identify tasks and task dependencies	Alliance web services, slave web services, goal dependencies and composition scenario.	Tasks and task dependencies.
4) Identify commitments	Tasks dependencies, scenario commitments description.	Commitments describing business relation.
5) Identify argumentative dialogues	Nonfunctional properties and commitments.	Accessitable commitments.

TABLE I
OUTLINE THE STEPS IN OUR METHODOLOGY.

*3) **Step 3: Identify Tasks and Dependencies**:* each role from step (2) has goal dependencies, then the ultimate objective of this step is to find task dependencies that will be responsible to achieve goals dependencies. Meanwhile, one goal (say g_1) may need a set of tasks (say $t_1 = \{t_{1.1}, t_{1.2}, \ldots\}$) to accomplish it. Subsequently, means-end analysis needs to identify this set of the tasks. Similarly, the task may be decomposed into subtasks and this decomposition analysis will iterate until no new task dependencies arise.

B. Architecture Design

The architectural design phase plays a crucial role in the design process. Initially, it defines the organization of the system in terms of the components and their interdependencies that are identified in previous phase. This step focuses on how system components work together to constitute a multiagent system and introduces resources, goals and roles as needed. This paper presents an architecture to tackle pitfalls of standard approaches that do not underpin business meaning and dynamic composition of existing services in which the components are agent-based web services and their interdependencies are specified in terms of commitments augmented with argumentation capabilities to reason about the validity of these commitments (we will explain commitment and argumentation later on). In fact, this architecture is a call and return style in the form of layered phases and an extension to the architecture we developed in [3], [4].

From Fig. 1 the main components of the proposed architecture are service providers, service consumers of web services, UDDI registers and communities with alliances structure. A community with alliances structure is organized dynamically according to the specifications discussed in previous phase (III-A). Hereafter we focus on operationalizing the steps through which the goal dependencies are to be fulfilled. The service providers publish and register the name of their services in UDDI registries with different nonfunctional proprieties (e.g., QoS) so that service consumers or users can search for appropriate QoS. More precisely, there are three kinds of agent-based web services ($master\text{-}ws$, $allaince\text{-}ws$ and $slave\text{-}ws$) constitute the structure of community and collaborate with each other to achieve users' requests.

A $master\text{-}ws$ can be implemented as a web service for compatibility purposes with the slave web services and alliances web services that populate the community as well. It delegates goal dependencies to $allaince\text{-}ws$ that will responsible for accomplishing them. The alliance web services manage micro-communities and decompose goal dependencies to slave web services that populate their alliance structure as well. Each $slave\text{-}ws$ signs up contract with its alliances to commit to satisfy the delegated subgoal. The slave web services collaborate with each other to achieve these goals. Of course each slave web service has the ability to delegate or assign incomplete tasks to other slave web services to complete its goal. Moreover, $allaince\text{-}ws$ can request from $master\text{-}ws$ to search for a new $slave\text{-}ws$ to join in its structure instead of the existing $slave\text{-}ws$ that it does not work well. When all slave web services satisfy their contracts, their alliance web services consequently achieve their composite contracts. Hence these alliance web services need to inform the $master\text{-}ws$ with results to finalize users' requests.

C. Detailed Design

This phase is intended to introduce additional details for each architectural component of a community structure. To support this phase, we adopt social commitment, argumentative dialogues and dialogue games protocol from the agent programming community.

*1) **Step 4: Identify Social Commitments**:* our methodology captures high-level business meaning via identifying commit-

ments between roles in terms of tasks. More precisely, this step analyzes each task dependency resulted from step (3) and that discovered in the architecture phase to identify the corresponding commitment. The notion here depends on the task dependency such that when the dependee (or the debtor) commits towards the depender (or the creditor) to execute the dependum task (e.g., t_1) or the commitment content, then a commitment exists. The condition of the commitment is defined by identifying a task that dependee needs to satisfy to perform commitment content. Meanwhile, the creditors need to verify some constraints that are conjuncted with this commitment to guide them to determine the corresponding tasks resulting from executing this commitment. Whereas, if the dependee is not committed towards the depender to execute a task, then no commitment exists, in this case the dependee executes the intrinsic task to achieve its internal goal.

*2) **Step 5: Identify Argumentative Dialogues**:* this step identifies arguments that can be used either in negotiation, persuasion or information seeking dialogue based on commitments identified in step (4).

Argumentative Locutions	Descriptions
Open-dialogue	A special argumentative act used to open the dialogue.
Accept($Ag\text{-}ws_2, C(id_x, \phi_x)$)	When $Ag\text{-}ws_2$ has an argument in favor of ϕ_x.
Refuse($Ag\text{-}ws_2, C(id_x, \phi_x)$)	When $Ag\text{-}ws_2$ has an argument against ϕ_x.
Attack($Ag\text{-}ws_2, C(id_x, \phi_x)$, $C(id_y, \phi_y)$)	When $Ag\text{-}ws_2$ attacks the content of $C(id_y, \phi_y)$ by the content of its commitment $C(id_x, \phi_x)$.
Challenge($Ag\text{-}ws_2, C(id_x, \phi_x)$)	When $Ag\text{-}ws_2$ has neither an argument for ϕ_x nor for $\neg\phi_x$, then it challenges ϕ_x.
Justify($Ag\text{-}ws_2, C(id_x, \phi_x)$, $C(id_y, \phi_y)$)	When $Ag\text{-}ws_1$ has a commitment $C(id_x, \phi_x)$ to justify another commitment $C(id_y, \phi_y)$.
Make-Offer($Ag\text{-}ws_1, C(id_x, \phi_x)$)	An $Ag\text{-}ws_1$ makes an offer ϕ_x to $Ag\text{-}ws_2$ when $Ag\text{-}ws_1$ has an argument in favor of ϕ_x.
Close-dialogue	A special argumentative act used to close the dialogue.

TABLE II
THE NATURAL DESCRIPTION OF THE ARGUMENTATIVE LOCUTIONS.

This methodology terms these arguments by *social arguments*, not only to emphasise their ability to resolve conflicts within a social community, but also to highlight the fact that two agent-based web services having task dependency can negotiate or persuade and, upon agreement, commit to each other for the specified value transfers. However, identifying such arguments is not merely the last step. Since agent-based web services need a language to express these arguments. In [4] we proposed *Horn logic language* to allow agent-based web services to express their arguments and to develop their reasoning capabilities within an argumentation system.

The formal specification of these arguments is defined in a dialogue game protocol, namely Persuasive-Negotiation Protocol for CWSs ($PNP\text{-}CWS$) [4]. We have eight argumentative locutions: {*Open, Accept, Refuse, Make-Offer, Attack, Challenge, Justify, Close*} forming the basic building blocks of this protocol. Here we present the description of these locutions in natural language (see Table (II)). To simplify the notation, a commitment will be denoted by $C(id_x, \phi_x)$ when the participating agents and the other elements are clear from the context. These locutions specify our communication language that can be used to communicate about satisfying goals or tasks.

***Definition** 1:* (Dialogue game) Let $Open(Ag\text{-}ws_1, C(id_k, Ag\text{-}ws_1, Ag\text{-}ws_2, C_x, \psi, \phi)$ be the opening action performed by $Ag\text{-}ws_1$ and sent to another agent $Ag\text{-}ws_2$ about content ϕ subject to ψ within alliance context C_x. A dialogue game Dg is a conjunction of rules, where the first rule defines the condition to enter Dg if the argumentation systems of $Ag\text{-}ws_1$ and $Ag\text{-}ws_2$ support the satisfaction of the commitment condition ψ. The other rules identify possible actions that an agent-based web service can use as a reply when receiving an action from another agent-based web service if a given condition $Cond_{ij}$ is satisfied. This conjunction is specified as follows:

$$
\begin{aligned}
Entry\ rule &: \mathsf{Open}(Ag\text{-}ws_1, C(id_k, Ag\text{-}ws_1, Ag\text{-}ws_2, C_x, \psi, \phi)) \\
&\quad \wedge Cond_k \\
Body\ rules &: \bigwedge\nolimits_{0<j\leq \mathbf{N}} \big(Action_i(Ag\text{-}ws_p, \mathsf{Ag\text{-}ws_m}, \mathsf{C(id_x}, \phi_\mathsf{x}), \\
&\qquad C(id_y, \phi_y)) \wedge Cond_{ij} \\
&\quad \Rightarrow \mathsf{Action}_{ij}(Ag\text{-}ws_n, \mathsf{Ag\text{-}ws_O}, \mathsf{C(id_z}, \phi_\mathsf{z}), C(id_w, \phi_w))\big)
\end{aligned}
$$

In the entry rule, the $Open$ action represents the opening of the dialogue game and is executed just one time at the beginning of the dialogue. The $Cond_k$ has two possibilities either the commitment condition ψ is true (i.e., can be generated from argumentation systems of $Ag\text{-}ws_1$ and $Ag\text{-}ws_2$) or false. The body rules are executed many times during the dialogue game. In these rules, the $Action_i(Ag\text{-}ws_p, Ag\text{-}ws_m, \mathsf{C(id_x}, \phi_\mathsf{x}), C(id_y, \phi_y))$ is an action of type i on the propositional commitment,since the commitment condition holds, where $Action_i \in \{Create, Fulfill, Violate, Release, Withdraw, Delegate, Assign, Accept, Refuse, Attack, Justify, Challenge, Make\text{-}Offer\}$. Moreover, the bold elements meaning that they could be removed. For example, if $Action_i \notin \{Delegate, Assign\}$, then the element $\mathsf{Ag\text{-}ws_m}$ could be removed. The $Action_{ij}(Ag\text{-}ws_n, \mathsf{Ag\text{-}ws_O}, \mathsf{C(id_z}, \phi_\mathsf{z}), C(id_w, \phi_w))$ is an action on the commitment of type j with content ϕ_w that depends on the action of type i, where $Action_{ij} \in \{Create, Fulfill, Violate, Release, Withdraw, Delegate, Assign, Accept, Refuse, Attack, Justify, Challenge, Make\text{-}Offer\}$. We notice that $n = w$ when $C(id_x, \phi_x)$ does exists (e.g., Attack and Justify), otherwise, $w = y$. We also have $n = m$ if $\mathsf{Ag\text{-}ws_m}$ does exist (e.g., Delegate and Assign), $\Rightarrow$ is the implication symbol for dialogue game rules, and $\mathbf{N}$ is the number of allowed actions that $Ag\text{-}ws_n$ can perform after receiving an action from $Ag\text{-}ws_p$ where $n, p \in \{1, 2\}$.

The commitment condition and content in this definition are Horn formulas, $Cond_k$ defines the possibility of entering

Dg when the condition of the commitment is satisfied. While $Cond_{ij}$ is expressed in terms of the possibility of generating an argument from the argumentation system.

To clarify the relationship between commitment and argumentation system, let us first define some notions used in our approach. The knowledge base of agent-based web service ($Ag\text{-}ws$) is denoted by $KB(Ag\text{-}ws)$, it contains all information related to functional and nonfunctional requirements of $Ag\text{-}ws$. The argumentation system of $Ag\text{-}ws$ is denoted by $Arg_{sys}(Ag\text{-}ws)$ using Horn logic language where $\phi \triangleleft Arg_{sys}(Ag\text{-}ws)$ denotes the fact that a Horn propositional formula ϕ can be generated from argumentation system of $Ag\text{-}ws$. While the formula $\neg\phi \triangleleft Arg_{sys}(Ag\text{-}ws)$ indicates that ϕ cannot be generated from argumentation system of $Ag\text{-}ws$.

The following is an example of dialogue game within alliance context C_x, in which after opening the dialogue, an alliance web service1 ($alliance\text{-}ws_1$) invites a slave web service1 ($slave\text{-}ws_1$) to join a current composition scenario where $alliance\text{-}ws_1$ knows that $slave\text{-}ws_1$ has a reasonable role to complete this scenario. The invitation is modeled using $Make\text{-}Offer$ locution. Then $slave\text{-}ws_1$ searches its $Arg_{sys}(slave\text{-}ws_1)$ to decide either to accept or refuse the invitation. $slave\text{-}ws_1$ accepts the invitation by verifying some constraints related to the performance of $alliance\text{-}ws_1$. That is, if the invited web service ($slave\text{-}ws_1$) has an argument favoring the received invitation (e.g., $alliance\text{-}ws_1$ has a good reputation) and does not have any argument against this invitation (e.g., $slave\text{-}ws_1$ does not commit to join any other alliance). Thereby $slave\text{-}ws_1$ has two choices either to accept this invitation, then $slave\text{-}ws_1$ creates a commitment towards $alliance\text{-}ws_1$ to join this composition scenario or to refuse, then $slave\text{-}ws_1$ releases from this commitment. The formal representation of the acceptance case of entering and accepting the invitation is as follows:

***Example** 1:*

1) Open$(alliance\text{-}ws_1, C(id_1, alliance\text{-}ws_1, slave\text{-}ws_1, \psi, \phi)) \wedge (\psi \triangleleft Arg_{sys}(slave\text{-}ws_1)) \wedge (\psi \triangleleft Arg_{sys}(alliance\text{-}ws_1))$
2) Make-Offer$(alliance\text{-}ws_1, C(id_1, \phi))$
$\wedge\ (Reputation\text{-}of\text{-}alliance = Good \triangleleft Arg_{sys}(slave\text{-}ws_1))$
$\wedge\ (\neg\ Commit\text{-}to\text{-}Join\text{-}an\text{-}alliance \triangleleft Arg_{sys}(slave\text{-}ws_1))$
$\Rightarrow$ Accept$(slave\text{-}ws_1, C(id_1, \phi))$
$where\ \psi = functionality\text{-}matches\text{-}composite\text{-}scenario$
$\phi = Invitation\text{-}for\text{-}joining$

The commitment manipulation underpins our approach with a simple mechanism of composite services between slave web services to handle users' requests through three-party actions (e.g., delegation, assignment). In simple case of delegation action (i.e., without metacommitment), we have two commitments among three agent-based web services where if an agent cannot able to complete its service, then it delegates the service to another agent (this delegation is made randomly). To clarify our notion about composing services, suppose $slave\text{-}ws_1$ is committed to $alliance\text{-}ws_1$ to bring about some facts within alliance context (C_x), if the condition ψ holds. Meanwhile, the argumentation systems of $slave\text{-}ws_1$ and $alliance\text{-}ws_1$ support this condition and the argumentation system of $slave\text{-}ws_1$ favors the content of this commitment. However, if any reason after creating the commitment, $slave\text{-}ws_1$ cannot complete it, then it will delegate the commitment to another agent-based web service in the same alliance (say, $slave\text{-}ws_2$). When $slave\text{-}ws_2$ observes the delegated commitment, it uses its argumentation system to search if it has an argument supporting the condition of this commitment. Also, if the argumentation system of $alliance\text{-}ws_1$ is still supporting the condition of this commitment, the $slave\text{-}ws_2$ will create a new commitment towards $alliance\text{-}ws_1$ (see Fig.2), formally:

***Example** 2:*

1) Open$(alliance\text{-}ws_1, C(id_1, alliance\text{-}ws_1, slave\text{-}ws_1, C_x, \psi, \phi)) \wedge (\psi \triangleleft Arg_{sys}(alliance\text{-}ws_1)) \wedge (\psi \triangleleft Arg_{sys}(slave\text{-}ws_1))$
2) Make-Offer$(alliance\text{-}ws_1, C(id_1, alliance\text{-}ws_1, slave\text{-}ws_1, \phi)) \wedge (\phi \triangleleft Arg_{sys}(slave\text{-}ws_1))$
$\Rightarrow$ Accept$(slave\text{-}ws_1, C(id_1, slave\text{-}ws_1, alliance\text{-}ws_1, \phi))$
3) Accept$(slave\text{-}ws_1, C(id_1, slave\text{-}ws_1, alliance\text{-}ws_1, \phi)) \wedge (\alpha \triangleleft Arg_{sys}(slave\text{-}ws_1))$
$\Rightarrow$ Delegate$(slave\text{-}ws_1, slave\text{-}ws_2, C(id_1, slave\text{-}ws_1, alliance\text{-}ws_1, C_x, \phi)) \wedge (\psi \triangleleft Arg_{sys}(alliance\text{-}ws_1)) \wedge (\psi \triangleleft Arg_{sys}(slave\text{-}ws_2))$
4) Delegate$(slave\text{-}ws_1, slave\text{-}ws_2, C(id_1, slave\text{-}ws_1, alliance\text{-}ws_1, C_x, \phi)) \wedge (\phi \triangleleft Arg_{sys}(slave\text{-}ws_2))$
$\Rightarrow$ Create$(slave\text{-}ws_2, C(id_2, slave\text{-}ws_2, alliance\text{-}ws_1, C_x, \phi))$
$where\ \psi = the\ condition\ of\ the\ commitment$
$\phi = bring\ about\ some\ facts$
$\alpha = \phi\ cannot\ be\ completed\ by\ slave\text{-}ws_1\ and$
$\phi\ can\ be\ performed\ by\ slave\text{-}ws_2$

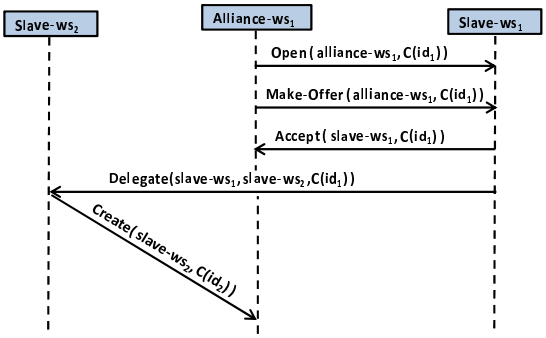

Fig. 2. The sequence diagram of the delegation action.

IV. Case Study

To illustrate the application of our methodology, we consider a real-life insurance claim processing that has been studied under the *CrossFlow* project [19] and presented in many works to manage business process (see [6], [15], [16]). Fig.3 shows the use case of this case study, which is about an

insurance company in Ireland (AGFIL) including the parties involved with their individual processes. AGFIL underwrites automobile insurance policies and covers losses incurred by customers. Europ Assist (EA) provides a 24-hour help-line service for receiving customer claims and assigns name of an approved repairer to customer. Lee CS is a consulting service that coordinates with AGFIL in receiving invoices and deals with repairers, adjustors and assessors to execute these claims. Moreover, AGFIL has the capability to decide if both a given claim is valid against fraud and payment will be sent to the repairer. Below the main steps of our methodology.

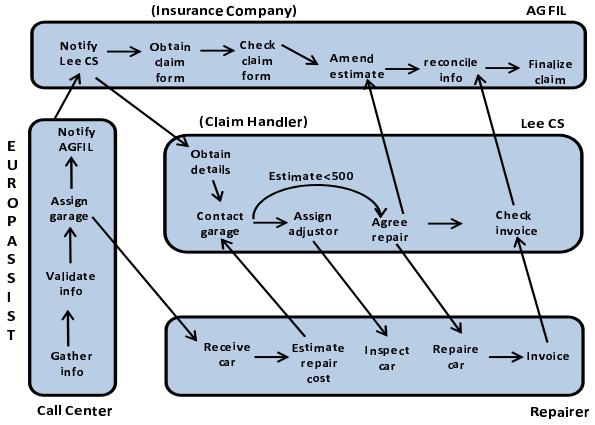

Fig. 3. Cross flow insurance claim processing [19].

A. Step 1

According to the ontology associated with the community, the functionality of the community is *InsuranceClaimProcessing* and this community includes all agent-based web services supporting the similar or part of this functionality. Thereafter, the engineering designer selects one of them as $master\text{-}ws_1$ (as the manager of the company) that starts to cluster the slave web services into alliances structure based on the policy classes (a form of nonfunctional properties related to QoS), such as $class_1$, $class_2$ and $class_3$ cover $100\%, 90\%, 80\%$ respectively from each claim request. We here only consider alliance structure that covers $QoS = \{100\%\}$ of the automobile damage and from now we refer to AGFIL by the Insurer (i.e., the role of this alliance is identified by the Insurer). The unique role names of slave web services that populate this alliance are defined as *call center* for Europ Assist and *assessor* for Lee CS, as well as *repairer* and *adjustor*. We modify this use case [16] by establishing a direct dependency between the Insurer on one side and repairer and adjustor on the other side. Also, we introduce the direct dependency between the customer and Insurer. At the end of this step, the Insure context is denoted by Ins_x and the $manager$ (or $master\text{-}ws_1$) delegates the customer's request to the Insurer.

B. Step 2

The customer has one relevant goal: *vehicle repair*, while the Insurer has the goals: *handle claim* and *maximize profits*. The last goal is represented in terms of soft goal. Thus the customer depends upon the Insurer to handle claim with $Qos = \{100\%\}$ and in exchange the Insurer depends upon the customer for paying the insurance premium [15]. Let us focus on such a goal, namely handle claim. The Insurer uses AND decomposition to decompose the goal ($G = \{handle\ claim\}$) into five subgoals, $G = \{claim\ reception, claim\ assessment, vehicle\ repair, claim\ finalization, vehicleinspection\}$. The Insurer delegates these subgoals to the respective roles within its alliance structure where the call center is responsible for the claim reception, the assessor for the claim assessment, the repairer for the vehicle repair and the adjustor for the vehicle inspection. The claim finalization will be performed by the assessor himself. By so doing, the Insurer pays service charge to each one of them after completing their goals.

C. Step 3

For the space limit reasons, we concentrate only on the call center, assessor and repairer task dependencies. The means-end analysis is used to identify tasks and task dependencies to each goal dependency from step (2). The claim reception goal, g_1, of the call center depends on four tasks to achieve it, $t_1 = \{gathering\ info, validating\ claim, assigning\ garage, sending\ claim\}$. The call center gathers information and assigns a garage when the customer reports an accident and validates claim information. Thus, the repairer depends on the call center to assign a garage and the customer depends on call center for gathering claim information. Meanwhile, the validate claim information task is decomposed into two subtasks, $t_{1.2} = \{request\ policy\ information, validate\ information\}$. According to the architecture design the call center can define a new goal to receive payment of claim reception charge from the Insurer via executing a task of *receiving payment* or delegate the task to another (e.g., the reporter to prepare its report). Finally, it sends a valid claim to the Insurer to *finalize claim* processing.

The assessor has claim assessment goal, g_2, delegated from the Insurer. The tasks needed to satisfy this goal are: $t_2 = \{receive\ claim, check\ invoice, agree\ to\ repair, obtain\ repair\ estimate, inspect\ vehicle\}$. The *receive claim* task depends upon *send claim* task of the Insurer to the assessor. Meanwhile, the Insurer and repairer depend on the assessors' tasks for *checking the invoice* and *agreeing to repair*. Moreover, the assessor depends upon the repairer to *obtain the repair estimate* by performing *estimation repair cost* task. The assessor depends upon the adjustor to *inspect a vehicle* and requires to define a new goal to *receive assessment fees* from the Insurer by executing a task of *receiving assessment fees*.

The repairer has a vehicle repair goal, g_3, and the tasks needed to satisfy this goal are denoted by $t_3 = \{repair\ vehicle, estimate\ repair\ cost, send\ invoice\}$. The customer depends upon the repairer for *repairing vehicle* when received *valid claim* from the customer. The assessor depends upon the repairer for *estimating repair cost* to decide

agreeing to repair or negotiating with the repairer. Meanwhile, the repairer depends on the assessor for *checking the invoice* and forwarding it to the Insurer, if the repairer sends the invoice. Likewise, the call center and the assessor, the repairer requires to define a new goal to *receive repair charge* from the Insurer by depending on executing a task of *receiving repair charge* [6], [16].

D. Step 4

This step transfers each task dependency into an appropriate commitment to represent business meaning of interacting parties of AGFIL. In [16] the authors have ignored the formulation of actions on commitments and focused only on commitment itself (although actions on commitments reflect dynamic behaviors of agents). For example, commitment $C(id_1, call\ center, customer, Ins_x, report\ accident \wedge valid\ claim, assign\ garage)$ means that when the customer reports an accident and if the claim is valid, then the call center commits to assign a garage to him within the Insurer. But, how the Insurer delegates claim reception to the call center, how the Insurer formally assigns the assessor to get the inspection fees from the adjustor, how the call center withdraws from his commitment, etc. Here we complement [16] with commitment operations. From step (2) the Insurer has five subgoals (claim reception, claim assessment,vehicle repair, claim finalization and vehicle inspection) that are delegated to the call center, assessor, repairer and adjustor respectively. Formally we define this delegation operation, but in the case of the Insurer that delegates claim reception to the call center only as follows:

$\mathsf{Delegate}(Insurer, call\ center, C(id_1, Insurer, Ins_x, customer, pay\ insurrance\ premium, claim\ reception))$

This operation intuitively means that the Insurer withdraws from the commitment and the call center creates a new commitment such that it becomes the debtor towards the customer to receive claim reception. Formally we need two steps to perform this:

$\mathsf{Withdraw}(Insurer, C(id_1, Insurer, policy\ holder, Ins_x, pay\ insurrance\ premium, claim\ reception)) \wedge$
$\mathsf{Create}(call\ center, C(id_{1.1}, call\ center, customer, Ins_x, claim\ reception))$

According to step (3) the claim reception needs four tasks to be achieved, one of them is gathering information from the customer. We define it formally as:

$\mathsf{Create}(call\ center, customer, C(id_{1.1}, call\ center, customer, Ins_x, report\ accident, gather\ info))$

Moreover, the Insurer assigns the assessor to obtain inspection fees from the adjustor when the estimate repair cost returned from repairer is more than 500 (a threshold amount).

$\mathsf{Assign}(Insurer, assessor, C(id_2, adjustor, Insurer, Ins_x, pay\ inspection\ fees, estimate\ inspection\ cost))$

This action is similar to the delegation action. Formally we need two steps to perform this:

$\mathsf{Release}(Insurer, C(id_2, adjustor, Insurer, Ins_x, pay\ inspection\ fees, estimate\ inspection\ cost)) \wedge$
$\mathsf{Create}(adjustor, C(id_{2.1}, assessor, Ins_x, pay\ inspection\ fees, estimate\ inspection\ cost))$

E. Step 5

We define two subscenarios from AGFIL scenario to explain why we need argumentative dialogues to reason about the validity of commitment operations. The first subscenario is established between the Insurer and assessor where the assessor commits to the Insurer to reach agreement with the repairer for the vehicle repair, formally: $C(id_3, assessor, Insurer, Ins_x, pay\ assess\ fees, agree\ to\ repair)$ [16]. However, the assessor cannot estimate his assessment fees without engaging in a dialogue with the repairer to reach a deal about "estimate the repair cost". The solution proposed in [16], which is based only on commitments, is not enough to specify this dialogue especially when the assessor needs to negotiate the repair charge with the repairer. The proposed argumentative dialogues are natural solutions to this problem. The following dialogue game explains only the steps after the commitment being delegated to the repairer. This means that *request to estimate repair cost* (i.e., ψ) is supported by argumentative systems of the assessor and repairer (i.e., $\psi \triangleleft Arg_{sys}(assessor) \wedge \ \psi \triangleleft Arg_{sys}(repairer)$). Then the dialogue is opened and this dialogue can be considered as a continuation to example (2). The repairer creates a commitment towards the assessor to estimate the repair cost. The assessor has a conflict with the Insurer because the estimated cost does not respect the delegated constrains from the Insurer (estimate cost should be < 500 to maximize the Insurer profits). Then assessor challenges the repairer to justify the estimated repair cost. When the argumentation system of the assessor supports the justification of the estimate repair cost, then it reaches a deal with the repairer and informs the Insurer to pay the assessed fees.

1) $\mathsf{Create}(repairer, C(id_{3.1}, repairer, assessor, Ins_x, \phi)) \wedge Cond_1$
   $\Rightarrow \mathsf{Challenge}(assessor, C(id_{3.1}, repairer, assessor, Ins_x, , \phi))$
2) $\mathsf{Challenge}(assessor, C(id_{3.1}, repairer, assessor, Ins_x, \phi)) \wedge Cond_2$
   $\Rightarrow \mathsf{Justify}(repairer, C(id_{3.2}, repairer, assessor, Ins_x, \phi'), C(id_{3.1}, repairer, assessor, Ins_x, \phi))$
3) $\mathsf{Justify}(repairer, C(id_{3.2}, repairer, assessor, Ins_x, \phi'), C(id_{3.1}, repairer, assessor, Ins_x, \phi)) \wedge Cond_3$
   $\Rightarrow \mathsf{Accept}(assessor, C(id_{3.2}, repairer, assessor, Ins_x, \phi'))$

$Where:$
$\phi = Estimate\text{-}Repair\text{-}Cost = V, with\ V\ is\ a\ given\ value$
$Cond_1 = \neg(\phi \triangleleft Arg_{sys}(assessor)) \wedge$
$\neg(\neg\phi \triangleleft Arg_{sys}(assessor))$,
$\phi' = \phi'_1 \wedge \phi'_2, Cond_2 = \phi' \triangleleft Arg_{sys}(repairer)$,
$Cond_3 = \phi' \triangleleft Arg_{sys}(assessor)$

Notice that the value of $\phi' = \phi'_1 \wedge \phi'_2$ means that the cost of repair includes the value of the $part_1$ (ϕ'_1) and $part_2$ (ϕ'_2).

Similarly, the adjustor and assessor can entre in negotiation dialogue about the estimated inspection cost. Moreover, argumentative dialogue in our case study can be used to spread agent-based web service knowledge in the form of information seeking dialogue. For example, when the call center requests policy information from the Insurer, the available information may be not enough to validate the received claim. Thus, the call center may need to spread his knowledge by requesting more information from the customer that enable the call center to send a valid information to the Insurer.

The second subscenario is established between the customer, Insurer and call center. When the customer reports an accident and his claim is valid, then the call center assigns a garage for vehicle repair. Otherwise, the call center does not assign a garage as it gathers invalid information from the customer compared to the information received from the Insurer. Thereby the intuitive semantic of the interaction between the call center and customer lacks clarity of business meaning related to the content of interaction (i.e, it is not meaningful). But, argumentative dialogue makes interaction more meaningful by guiding the call center to undertake a subtle decisions when the conflicts arise, then it challenges the customer to justify the validity of its commitment and to reach a mutual agrement that helps the customer to repair the vehicle. Moreover, the Insurer should persuade the call center, repairer, assessor and adjustor to join alliance structure by offering rewards that alliance grants to them.

V. Related Work and Conclusions

In this paper, we first exposed some obstacles that restrict the use of agent-based web services in complex business applications and then, offered some solutions to tackle these obstacles through extending community structure with alliances structure and proposing a new methodology based on Tropos methodology. This methodology synthesizes three approaches: mentalistic states, social commitments and argumentative dialogues. The mentalistic artifacts such as goals, tasks and dependencies modeling techniques are compatible with concepts of Tropos (e.g., plan in Tropos is the task here). The social commitments define dependencies between agent-based web services in the early stages as business-meaning of interactions and guide Tropos for accommodating and tracing the changes in the various phases of the methodology. While Tropos provides commitment with cues that identify different parties of dependencies (e.g., agents, goals, plans). Moreover, commitments enable the community to support dynamic reconfiguration of business interactions via delegation and assignment operations without altering the overarching structure of web services interactions. Argumentative dialogues support web services with richer capabilities to negotiate or seek information with other peers via reasoning about the correctness of commitments

The social-arguments proposed here are different from the approach proposed in [8] for argumentation-based negotiation based on social commitments. The agents are influenced by social relationships and negotiate with other peers based on their organizational roles. The approach lists the possible rules that can be applied when conflicts of interest occur. These rules, for example, are used to reject or accept proposals as well as to enforce the social relations within a multiagent system. However, the rules in [8] are not used for justifying the content of commitments within negotiation dialogue and they do not consider agents' goals that can affect agents decisions as we have done here.

Let us now focus on comparing the proposed methodology with the related ones. Many of Agent-Oriented Software Engineering (AOSE) methodologies have been proposed over the last years based on the concepts of actors, roles, goals and plans include *Gaia, Prometheus, MaSE, and Tropos*. These methodologies support various phases of the software development life cycle, but Tropos differs from them in including an early requirements phase. Moreover, some of these methodologies like Gaia differs from Tropos in involving safety and liveness conditions for the processes and agents should be coarse-grained computational systems (like UNIX process). Meanwhile, Gaia lacks reasoning scheme based on early requirements engineering which limits the flexibility of Gaia, as well as implementation phase is not covered in this methodology. Our methodology complements these methodologies by concentrating on social arguments and argumentative dialogues, which they are ignored in these methodologies. A key difference between our methodology and Tropos [17] is the considerations of concepts of community, commitment and argumentative dialogue. In Tropos the depender depends upon the dependee for achieving a goal or executing a plan without any condition from the depender's side, but in commitment the debtor is obliged towards the creditor to bring about its commitment when a condition is hold. Thus, Tropos lacks capability to model real-life business scenarios between economic partners [15].

Pankaj et al. [15] enhanced Tropos methodology with concept of commitments to capture business meaning of interactions among independent parties in early requirement phases and successively refined using means-end and AND/OR analysis during the progress of the system being engineered. This methodology [15] is close to our methodology but it lacks the argumentative capabilities to enable agent-based web services to negotiate or seek information with other peers based on their commitments. Moreover, the authors translate goals into concrete tasks for their fulfillment before considering the overall organization of the system, which makes software systems fragile and less reusable although commitments can be modified. The reason is that they ignore architecture design phase. We present the architecture design phase with alliances structure to satisfy a number of quality requirements related to performance, usability, modifiability and reusability. For example, the Insurer can delegate the responsibility to the assessor to finalize users' claims or to pay inspection fees to the adjustor on his behalf.

Desai et al. [7] proposed Amoeba as a process modeling methodology based on commitment protocols. This methodology guides software designers to evolve requirements based

on separating service interactions into two layers: commitment protocols and polices. They represent business process in terms of fine-grained messages with commitments where commitments capture business interactions. Our methodology includes in addition of that mentalistic states and argumentative dialogues.

Riemsdijk et al. [18] used a goal-oriented approach inspired by the field of cognitive agent programming for service-oriented computing to help handling failures and specify the semantics of the services. An orchestration approach has been used to coordinate the invocations of services from a workflow. Burmeister et al. [5] presented an approach to business process management using BDI-agent features to capture the ability of the process to adapt and pro-actively adapt itself to a changing environment (or what they call an "agile process") to avoid problems before they arise. The goal-oriented approach is the core of our methodology, but it differs from [5], [18] in considering goals as commitments, which provide more flexibility in terms of manipulation (e.g., assign, release or withdraw) and agents use argumentative dialogues to enable each participant to satisfy its hard goals and soft goals in an efficient manner. Moreover, an orchestration reflects only one participant's view of the overall business process and lacks business meaning of service engagements. The conceptual framework proposed in [5] has been successfully applied to a business process for engineering change management domain, but it lacks the cooperation between agents.

Li et al. [9] proposed an agent-based framework to model and develop dynamic service-oriented operations. In our approach, web services are also viewed as software agents, but what is new in our approach is that web services can communicate with other peers within a community by dialogue game protocols.

Regarding to CWSs, Maamar et al. [11] recently presented an engineering methodology for modeling CWSs based on the concepts that assist in using community, selecting web services, identifying allowable operations to each web service and deploying community. However, they ignore the collaboration between web services, high-level business meaning of interactions, alliance structure, user interactions (although these interactions are crucial and ought to be recognized within community). Last but not least, Medjahed et al. [14] proposed the WebBIS system as a generic framework for composing and managing web services in terms of pull-and push-communities within dynamic environments. Our methodology underpins a simple mechanism for empowering composition of web services based on means-end and AND/OR analysis within the notion of delegation operation that captures flexible interactions.

As future work, we aim to formalize the strategic polices of service providers that help organizing web services in alliances structure and investigating the organization laws and norms ruling the roles within the community. The trustworthiness level of a master web service towards alliance web services and how a social-argumentative dialogue model is automatically mapped into a BDI-agent specification that can, at execution time, give useful feedback to refine the original design are fundamental issues we plan to investigate.

ACKNOWLEDGEMENTS

The authors would like to thank the reviewers for their valuable comments and suggestions. They also would like to thank NSERC (Canada), NATEQ FQRSC (Québec) and ERESSON for their financial support.

REFERENCES

[1] B. Benatallah, M. Dumas and Q.Z. Sheng. Facilitating the Rapid Development and Scalable Orchestration of Composite Web Services. J Distrib Parallel Databases vol.17(1), pp.5-37, 2005.

[2] B. Benatallah, Q.Z. Sheng and M. Dumas. The Self-Serv Environment for Web Services Composition. IEEE Internet Computing, vol.7(1), pp.40-48, 2003.

[3] J. Bentahar, Z. Maamar, D. Benslimane and P. Thiran. An Argumentation Framework for Communities of Web Services. IEEE Intel. Sys. vol.22(6), pp.75-83, 2007.

[4] J. Bentahar, Z. Maamar, W. Wan, D. Benslimane, P. Thiran and S. Subramanian. Agent-Based Communities of Web Services: An Argumentation-Driven Approach. In Service Oriented Computing and Applications, Vol.2(4), pp.219-238, 2008, Springer.

[5] B. Burmeister, M. Arnold, F. Copaciu, G. Rimassa. BDI-Agents for Agile Goal-Oriented Business Processes. In Proc. of 7th Int. Conf. on Aut. Agents and Multiagent Sys. (AAMAS 2008), pp.37-44, 2008.

[6] N. Desai, A.K. Chopra and M.P. Singh. Business Process Adaptations via Protocols. In Proc. of the IEEE Int. Conf. on Services Computing (SCC), pp.103-110, 2006.

[7] N. Desai, A.K. Chopra and M.P. Singh. Amoeba: A Methodology for Modeling and Evolution of Cross-Organizational Business Processes. ACM Transactions on Software Eng. and Methodology (TOSEM), 2009.

[8] N.C. Karunatillake, N.R. Jennings, I. Rahwan and T.J. Norman. Argument-Based Negotiation in a Social Context. In Proc. AAMAS Workshop on Argumentation, pp.74-88, May 2005.

[9] Y. Li, W. Shen and H. Chenniwa. Agent-Based Web Services Framework and Development Environment. Comput Intell, vol.20(4), 2004.

[10] Z. Maamar, M. Lahkim, D. Benslimane, P. Thiran and S. Sattanathan. Web Services Communities-Concepts and Operations. In Proc. of the 3rd int. conf. on web information sys. and technologies (WEBIST'2007), Barcelona.

[11] Z. Maamar, S. Subramanian, J. Bentahar, P. Thiran P and D. Benslimane. An Approach to Engineer Communities of Web Services Concepts, Architecture, Operation, and Deployment. In the Int. Journal of E-Business Research, vol.5(4), 2009, IGI Global.

[12] B. Medjahed and Y. Atif. Context-Based Matching for Web Service Composition. Distrib Parallel Databases. Springer, Heidelberg, vol.21(1), pp.5-37, 2007.

[13] B. Medjahed and B. Bouguettaya. A Dynamic Foundational Architecture for Semantic Web Services. Distributed and Parallel Databases. Kluwer, Dordrecht, vol.17(2), pp.179-206, 2005.

[14] B. Medjahed, B. Bouguettaya and A. Elmagarmid. WebBIS: An Infrastucture for Agile Integration of Web Services. Int. J. Cooperative Inf. Syst. (IJCIS) , vol.13(2), pp.121-158, 2004.

[15] P.R. Telang and M.P. Singh. Enhancing Tropos with Commitments: A business Metamodel and Methodology. Alex Borgida, Vinay Chaudhri, Paolo Giorgini and Eric Yu (eds), Conceptual Modeling: Foundations and Applications, June 2009.

[16] P.R. Telang and M.P. Singh. Business Modeling via Commitments. In Proc. of the 7th AAMAS Workshop on Service-Oriented Computing: Agents, Semantics and Eng. (SOCASE), May 2009.

[17] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia and J. Mylopoulos. Tropos: An Agent-Oriented Software Development Methodology. Autonomous Agents and Multi-Agent Systems, vol.8(3), pp.203-236, 2004.

[18] M.B. van Riemsdijk, M. Wirsing. Using Goals for Flexible Service Orchestration: A First Step. In Service-Oriented Computing: Agents, Semantics, and Eng. vol.(4504) of LNCS, pp. 31-48. 2007, Springer.

[19] S. Browne and M. Kellett. Insurance (Motor Damage Claims) Scenario. Document Identifier D1.a, CrossFlow Consortium, 1999.

[20] M.P. Singh. An Ontology for Commitments in Multiagent Systems: Toward A unification of Normative Concepts. AI and Law, vol.7, pp.97-113, 1999.