Modelling and Solving QoS Composition Problem Using Fuzzy DisCSP

Xuan Thang Nguyen
Faculty of Information and Communication Technologies
Swinburne University of Technology
Melbourne VIC 3122, Australia
Email:xnguyen@ict.swin.edu.au

Ryszard Kowalczyk
Faculty of Information and Communication Technologies
Swinburne University of Technology
Melbourne VIC 3122, Australia
Email: rkowalczyk@ict.swin.edu.au

Manh Tan Phan
Faculty of Information and Communication Technologies
Swinburne University of Technology
Melbourne VIC 3122, Australia
Email: tphan@ict.swin.edu.au

***Abstract*— Web service compositions have attracted considerable efforts in the context of supporting enterprise application integrations. For a composite service, in addition to its functional requirements, QoS requirements are important and deserve a special attention. The central question to a QoS composition problem is how to compose a service from different subcomponent services so that its overall QoS can satisfy certain requirements. In this paper, we propose an agent-based method using Fuzzy Distributed Constraint Satisfaction Problem (Fuzzy DisCSP) techniques to solve this problem. We show that by using the composition structures, local constraints can be constructed and used with DisCSP. We also present an a new algorithm called the Fuzzy constraint satisfaction Algorithm for Distributed Environment (FADE) to solve the problem and discuss our experiment in building a prototypical system to prove the feasibility of our approach.**

I. Introduction

There have been tremendous developments in the area of Web services recently. Web services' capability for integrating distributed and heterogeneous applications across organizational boundaries has attracted considerable research in various areas including Web service discovery, composition, and management. Web service composition focuses on the problem of composing a value-added composite Web service from different Web services. There are two key requirements in in the composition process: finding syntactic and semantic matches of the component Web services and satisfying non-functional requirements or Quality of Service (QoS). We call the latter - QoS composition problem for composite Web services as opposed to the former - functionality composition problem. Because component Web services of a composite Web service can also be composite, a composite Web service can have any number of composition nested level. In parallel to the advancement of Web services, the MAS and AI community has shown an increasing interest in the Distributed Constraint Satisfaction Problem (DisCSP) in the last few years. DisCSP has been widely viewed as a powerful paradigm for solving combinatorial problems arising in distributed, multi-agent environments. A DisCSP is a problem with finite number of variables, each of which has a finite and discrete set of possible values and a set of constraints over the variables. These variables and constraints are distributed among a set of autonomous and communicating agents. A solution in DisCSP is an instantiation of all variables such that all the constraints are satisfied. Fuzzy DisCSP is a DisCSP with soft constraints, i.e. different constraints have different important levels. In negotiation context, fuzzy constraints have been used widely to model service clients and providers' preferences over attribute values of products. Fuzzy constraints are useful to describe the preferences or utility functions of Web service providers. In this paper we propose a Fuzzy DisCSP-based technique to solve the QoS composition problem. Our algorithm models providers' preferences as fuzzy constraints and supports the notions of preference priorities, i.e different providers may have different priorities of being satisfied. The rest of the paper is organized as follows. A literature review on Web service compositions and DisCSP are presented in the next section. In Section 3, we model the QoS composition problem as a DisCSP. Section 4 proposes an algorithm to construct constraints from a Web service composition structure. An enhancement of the Asynchronous Aggregate Search (AAS) algorithm, which can handle multiple variables in each agent with a new heuristic developed to exploit QoS parameters' characteristics, is presented in Section 5. Section 6 discusses our initial experiment in building a prototypical system as a proof of concept for our proposal. Finally, conclusions and future work are discussed in Section 7.

II. Related Work

There have been several studies on Web service compositions. A survey of various studies on functionality compositions can be found in [17]. QoS compositions are discussed in [9], [1], [7], [5]. In [12], numeric calculations of execution time and cost of a composite service are presented. In [7], the authors discuss a method for QoS aggregation based on Web service composition patterns [19]. More related work on QoS planning can be found in [9], in which the authors propose an approach for selecting optimal sub-providers from a list of service providers. While these research focus on functional and QoS compositions at the same time, they solve the composition problems at each provider independently. Consequently, the real complexity of composing Web services which lead to many nested compositions or have may dependencies is

IEEE International Conference on Web Services (ICWS'06)
0-7695-2669-1/06 $20.00 © 2006 IEEE

overlooked. Also, these works do not make use of possible collaborations between different providers in a composition. This collaboration is proven to be important as the new Web service choreography stardard of WS-CDL [21] become more matured.

To overcome this limitation, we propose to use Fuzzy DisCSP techniques in which service providers from different nested levels in a composition can collaborate to solve the QoS composition problem together. To our best knowledge, there has been no application of DisCSP in this area so far. DisCSP is a major technique for coordination and conflict resolutions in a distributed and collaborative environment. A DisCSP is a constraint satisfaction problem in which the variables and constraints on these variables are distributed among independent but communicating agents. One important motivation behind the DisCSP paradigm is that it allows agents to keep their constraints privately while permits the solution to be solved globally. Many studies have been done in proposing algorithms for solving DisCSPs, such as Asynchronous Backtracking (ABT) [23], Asynchronous Weak-Commitment (AWC) search [22], and lately Asynchronous Distributed constraint Optimization with quality guarantees (ADOPT) [13], [2] to name a few. ABT is normally considered as the base algorithm for others.

Fuzzy DisCSP has lately captured the interest of MAS community in the context of negotiation. In [8], utility theory and fuzzy constraint-based are employed for the negotiation process. These works focus on bilateral negotiations and when many agents take part, a central coordinating agent may be required. The work in [10] is interesting as it promotes a framework for multi-lateral negotiation with a rotating central coordinating agent. However it does not take advantage of the current advancement of DisCSP. Also, even though there has been substantial research on different levels of importance of constraints in the Priority Fuzzy Constraint Satisfaction Problem(PFCSP) framework [11], little work has been done on priorities of agents in terms of satisfaction levels. In practice, different Web service providers may have different priorities for their satisfaction levels to be honored. For example, a composite Web service provider (main contractor) would naturally has higher priority than a component Web service providers (sub contractors). In [6], an iterative approach is used to solve a Fuzzy DisCSP through as a series of DisCSP is presented. This algorithm shares some similarity with our work. This algorithm is one among a few available algorithms for solving Fuzzy DisCSP. However the algorithm also does not address the satisfaction priorities. Because the algorithm can be extended to multivariables and n-ary constraints, we will use it to compare against our algorithm.

Util now, two popular problems in the DisCSP application domain are Meeting Scheduling and SensorCSP [3]. In the Meeting Scheduling problem, multiple agents negotiate to find a meeting time that can satisfy all agents' personal constraints. In the SensorCSP there are a number of sensors modeled as agents and targets where a target is tracked if k sensors are tracking it at the same time. The sensors have to cooperate to track all the targets with their own constraints that a sensor can only track one target at a time. In this paper, we introduce another problem that can be modeled and solved with DisCSP - the problem of QoS composition. This problem is of a great interest to the agent community.

III. Modeling QoS Composition Problem as DisCSP

The DisCSP and Fuzzy DisCSP can be formally defined as follows.

Definition 1: A Distributed Constraint Satisfaction Problem $\mathbf{P}=\langle \mathbf{X},\mathbf{D},\mathbf{C},\mathbf{N}\rangle$ is a problem defined on a set of variables $\mathbf{X}=\{X_1,...,X_n\}$, a discrete finite domains for each of the variables $\mathbf{D}=\{\mathbf{D}_1,...,\mathbf{D}_n\}$, and a set of constraints $\mathbf{C}=\{C_1,...,C_m\}$ on possible values of variables. These variables and constraints are distributed among a set of agent $\mathbf{A}=\{A_1,...,A_k\}$. If an agent A_l holds a constraint C_q, it also must hold all variables contained in C_q. A solution for **P** is an assignment of variables in **X** from their domains so that all constraints in **C** are satisfied.

In some DisCSP framework like the one introduced with ABT [23], the definition goes further by assigning owners (agents) to variables. Proposals on a variable can be made only by the owner of that variable. Slightly different frameworks have been used in the literature [16] where agent owns problems and shares variables. In this paper, we use the second approach as it is more flexible to model and solve problems with multi variables.

Definition 2: A Fuzzy Distributed Constraint Satisfaction $\mathbf{P}^f$ is an extension of a DisCSP $\mathbf{P} = \langle \mathbf{X},\mathbf{D},\mathbf{C}^f,\mathbf{N}\rangle$ in which a constraint C_j^f, j=1,...,m on a subset of the variables $\{X_{j1},...,X_{jk}\}\subseteq$ X represents a fuzzy relation defined on the Cartesian product space $\mathbf{D}_{j1} \times ... \times \mathbf{D}_{jk}$ characterized by a membership function $\mu_{C_j^f}$. That is

$\mu_{C_j^f}$: $\prod_{p=1}^{k} \mathbf{D}_{jp} \rightarrow$ [0,1], j=1,...,m.

The global satisfaction degree of an assignment x_S of all variable in **X** is defined as:

$sat(x_S) = \oplus\{\ \mu_{C_j^f}(x_S{\downarrow}C_j^f) \mid C_j^f \in \mathbf{C}^f\}$

in which $x_S{\downarrow}C_j^f$ is the projection of x_S into variable space of C_j^f or the part of assignment in x_S for variables of C_j^f. $\oplus$ is a T-norm operator and commonly used as the *min* fuction. In general, x_S is a solution of $\mathbf{P}^f$ if it maximinzes the global satisfaction degree, or it has the global satisfaction degree greater than or equal to a predefined threshold value. For each assignment, we define the agent satisfaction level for an agent A_n is the satisfaction degree of the assignment on the constraints known by A_n, i.e.

$sat_{A_n}(x_S) = \oplus\{\ \mu_{C_j^f}(x_S{\downarrow}C_j^f) \mid C_j^f \in \mathbf{C}^f,\ C_j^f$ held by $A_n\}$

In the QoS composition problem, multiple service providers participate in the service composition process. Each provider has its own constraints on QoS levels of some parameters. The QoS levels may be continuous but in most cases the service providers and consumers are only interested in discrete values of the QoS levels; for example, discrete values of cost, disk space and response time. Some constraints of the providers

can be revealed, for examples through different advertised *classes of service* and others must be completely private such as resource limitations, business rules, organizational policies and service composition structures. The goal of the QoS composition problem is to find a solution to all providers so that every providers' constraints are satisfied. In addition, each provider has a different level of satisfaction on a solution. Therefore, it is desireable to maximize the global satisfaction level for all providers.

Essentially, a QoS service composition problem is a combinatoral problem and can be modeled in DisCSP framework. In this model each service provider can be considered as an agent (an autonomously processing entity) in a constraint network. Each QoS parameter is mapped into a variable in the constraint network; and the set of providers' constraints is mapped into the network's constraint set. The problem of maximizing the global satisfaction level is equivalent to the constraint network with fuzzified constraints.

To facilitate the discussion, we start by a simple example without preferences. Figure 1 shows an example of five agents A_1, A_2, A_3, A_4, A_5 together forming an Attraction Service composite service. Each agent has its own constraints on QoS.

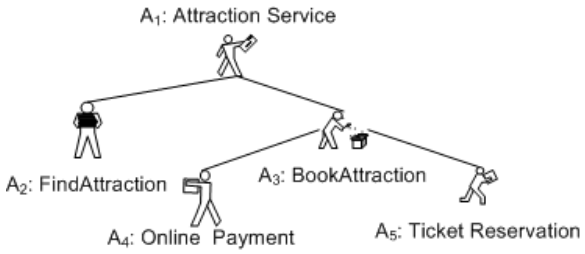

Fig. 1. A scenario of a composite Attraction Service which has a nested composition structure with the Book Attraction service offered by A_3.

In Figure 2, c(S) and t(S) define the total cost and response time of a composite service S while $c_{i:own}$ and $t_{i:own}$ define the cost and response time introduced by the agent A_i itself. The c variable can take a price between \$1 and \$100 (USD) and the t variable can take an integer value between 1 and 100 (ms) for all services. In a more complex example, to negotiate on the values of cost and response time, A_3 and A_4 may need to negotiate on other variables as well. In other words, the set of variables at each agent may not represent the same set of QoS parameters. Also, the example in Figure 1 gives an impression of a tree structure. However, this is not always true as many consumers can use the same session of a Web service.

To show the differences between not applying and applying DisCSP techniques, we consider the two most commonly used approaches for QoS composition problems [9], [1], [7], [5]:

- In the first approach, QoS composition is solved incrementally. Firstly, A_1 negotiates with A_2 and A_3 for attraction finding and booking. A_2 agrees to provide a *FindAttraction* service with, for example, {1ms, \$10} for {response time, cost}. A_3 agrees to provide its *BookAttraction* service with, for example, {4 ms, \$10} because it is confident to find sub-providers to support these values. After that, A_3 contacts and negotiates with A_4 and A_5. If A_4 agrees on {1ms, \$5} and A_5 agrees on {3ms, \$5} for their services respectively, a solution is found. Otherwise, A_3 have to find a substitute of A_4 and A_5.
- In the second approach, synchronous backtracking is used to solve the QoS composition problem. A1 proposes some cost and response time values to A_2 and A_3. For example, it proposes {1ms, \$15} to A_2 and {4ms, \$5} to A_3. A_2 accepts the proposal while A_3 subsequently negotiates with A_4 and A_5 before responding to A_1. In the negotiation, if A_4 and A_5 cannot satisfy with any proposals from A_3; A_3 backtracks to A_1 with a refusal on {4ms, 5 USD} values. A1 then has to negotiate for another value with A_2 and A_3. The above steps are repeated until A_4 and A_5 can agree on some proposals from A_3.

A_1: Attraction Service
c(AttractionService) =
c(FindAttraction) + c(BookAttraction)+ $c_{1:own}$
t(AttractionService) =
t(FindAttraction) + t(BookAttraction)+$t_{1:own}$
c(AttractionService) $<$ \$20
t(AttractionService) $<$ 5ms
A_2: Find Attraction
$c_{2:own}$ $>$ \$5
$t_{2:own}$ $>$ 1ms
$c_{2:own}$ = \$5 + (4ms -$t_{2:own}$) x \$1/ms
A_3: Book Attraction
c(BookAttraction)=
c(OnlinePayment) +c(TicketReservation) +$c_{3:own}$
t(BookAttraction)=
t(OnlinePayment) +t(TicketReservation) +$t_{3:own}$
A_4: Online Payment
$c_{4:own}$ $>$ \$2
$t_{4:own}$ $>$ 1ms
A_5: Ticket Reservation
$c_{5:own}$ $>$ \$3
$t_{5:own}$ $>$ 1 ms

Fig. 2. Illustration of different constraints held by each agent

It can be seen that both approaches are synchronous, i.e. each agent has to wait for responses from other agents before it can proceed. Also, each agent is only aware of its local composition problem which is formed whenever a request from a client arrives. For example, A_3 is only aware of a composition problem when it receives a proposal from A_1. To address this limitation, our approach allows every agent to be aware of the global QoS requirement, and encourages them to take part in the solving process asynchronously as soon as possible after a conceptual functional composition is formed

and before and contracts are singed. In the functionality composition phase, when an agent receives a problem *id* from its consumers, it generates a new problem *id* if necessary, and then forwards this to its providers to inform them that there is a global QoS problem which needs to be solved. In particular, a problem *id* received in combination with the sender address creates a unique context to identify a particular global QoS problem.

IV. Constraint Deduction from Composition Structure

As stated in the previous part, constraints can be formed from different items such as business rules, organizational policies, or composition structures. Translations from business rules and organizational policies into QoS constraints may vary from organizations to another since these rules and policies can be represented and interpreted differently. In this section, we show how constraints can be formulated from the composite structures of services offered at each agent. For examples, how the first two constraints in A_1 and A_3 in Figure 2 were constructed. These constraints represent the relationships among QoS parameters of a composite service and of its component services. Such constraint formations and deductions are useful if new QoS compositions need to be done on an existing functional composition. In practice, this happens because a Web service may expect to have new customers and hence can have requests of new QoS classes. In this paper, we use the term "QoS constraint formulation" to describe the process of finding these constraints/relationships from a composition input. The composition is presented in BPEL4WS (the Business Process Execution Language for Web services) format. BPEL4WS is selected because it is a popular Web service composition language supported by different Web service vendors. Our approach bases on important work in [19], [18], [7]. In [19] and [18], the authors conduct a survey and collect a set of workflow patterns which have been used in workflow languages today, including Web service composition languages. In [7] the authors map these workflow patterns into QoS composition patterns. We combine these results to form a mapping from BPEL4WS into QoS composition patterns as shown in Table 1. It is worth noting that Table 1 only presents composition patterns supported by BPEL4WS. Our QoS constraint formulation process consists of the following three main steps:

1) Construct constraint formulas for QoS composition patterns: Sequence, AND (Split/Join), XOR (Split/Join) and OR (Split/Join).
2) Iteratively decompose the composite structure through BPEL workflow patterns into smaller sub-compositions until this cannot be done any further.
3) Iteratively re-construct the constraints in a composition from constraints of its sub-compositions. The reconstruction uses formulas available in step 1 above.

In reference to other efforts on QoS aggregation [7], [9], [5], [12], our constraints can be considered as means to compute the QoS aggregations. The difference is that we construct the formulas to aggregate QoS instead of compute a value. Formulas for simple composition patterns in step 1 with different QoS attributes can be found in [7]. Table 2 presents a partial list of these. It is important to know that our algorithm does not restrict to the forumals presented in [7].

No.	Workflow Pattern (BPEL pattern)	QoS Composition Pattern
Basic Control Flow Patterns		
1	Sequence (Sequence)	Sequence
2	Parallel Split (Flow/Link)	AND Split
3	Synchronization (Flow/Link)	AND Join
4	Exclusive Choice (Switch/Link)	XOR Split
5	Simple Merge (Switch/Link)	XOR Join
Advanced Branching and Synchronization Patterns		
6	Multi-choice (Link)	OR Split
Structural Patterns		
7	Implicit Termination (By Default)	
Patterns Involving Multiple Instances		
8	M.I. Without Synchronization(Flow)	AND Split and AND Join
State Based Patterns		
9	Deferred Choice(Pick)	XOR Split
10	Interleaved Parallel (Serializable Scope)	Sequence
11	Cancel Activity (Terminate)	

TABLE I
WORKFLOW PATTERNS TO QOS COMPOSITION PATTERNS

	Sequence	AND Split/Join	OR Split/Join
Cost	Sum	Sum	Max
Response Time	Sum	Max	Min

TABLE II
FORMULAS OF CONSTRAINTS FOR SIMPLE COMPOSITION PATTERNS OF SEQUENCE, AND AND OR

```
CONSTRUCT-CONSTRAINT(qos-name, activity )
as XPath
1 activity-type ←get type of the activity
2 if activity-type ∈
{"assign", "throw", "wait",
"empty", "scope", "compensate", "terminate"}
3 return null
4 if activity type ∈{ "invoke", "receive"}
5 service-name ← get name of the service
invoked by activity
6 return Xpath(service-name)
7 sub-activities←get component activities from activity
8 qos-pattern-name←get the QoS
composition pattern equivalent to activity
9 Xpath-array ← {CONSTRUCT-CONSTRAINT(
qos-name, sub-activities[i])}
10 return F(qos-name, qos-pattern-name, Xpath-array)
```

Fig. 3. Constraint formation algorithm

IEEE International Conference on Web Services (ICWS'06)
0-7695-2669-1/06 $20.00 © 2006 IEEE

```
<flow name= "Shipping">
<sequence>
<invoke name= "ShipmentAir"/>
</sequence>
<sequence>
<invoke name= "ShipmentWater"/>
</sequence>
<sequence>
<invoke name= "ShipmentLand"/>
</sequence>
</flow>
```

Fig. 4. An example of BPEL activity

These formulas can be customized to suit particular scenarios and QoS characteristics.

For a more detailed explanation, the pseudo-code for our constraint formulation process is listed in Figure 3. In our algorithm, the BPEL4WS composition is represented as a *ProcessDef* object which has a base *Activity* object. The *Activity* class represents all possible activities available in BPEL4WS processes such as "receive", "reply", "invoke", "sequence" and "switch". Hence, an *Activity* object is composed of other *Activity* objects. XPath class represents an XPATH expression. The main idea behinds the *Construct-Constraint* algorithm in Figure 3 is to iteratively decompose a BPEL activity and apply equivalent QoS composition formulas (table 2) at each iteration. New variables are added whenever "invoke" and "receive" activities are encountered. In the line 10, F is a constraint function (constructed from Table 2) for QoS composition patterns. For an example, a composite service with a structure listed in Figure IV after passing through the *Construct-Constraint* function will produce t(Shipping)= max(t(ShipmentAir), t(ShipmentWater), t(ShipmentLand)) for the input of *qos-name* as *response time*.

V. Solving the QoS composition problem with DisCSP algorithms

A. The ADE algorithm

There have recently been many publications on DisCSP algorithms. Traditionally these algorithms are developed and demonstrated in the context of the Meeting Scheduling and Sensor Network problems as discussed at the beginning of this paper. Often the techniques used in these algorithms combine tree-search with backtracking, look-ahead, and back-jumping. However, there are some characteristics that make the QoS composition problem different from the Meeting Scheduling and Sensor Network problems:

- Each agent holds more than one variable.
- Each agent holds a set of interested QoS parameters and the variables that represent these parameters.
- Local constraints in QoS problem can be very complex.
- Provider agents are not willing to reveal their consumer agents' addresses to others.

process-ok($<x_j,s_j,h_j>$) **do**
1. **if**(history(x_j) invalidates h_j) return;
2. add $<x_j,s_j,h_j>$ to agent-view
3. update nogood list store
4. **check-agent-view**

process-nogood(A_j,$\neg$N) **do**
1. update agent-view for any new assignments in $\neg$N
2. send setup channel request to A_j

not connected agents in A^- which
hold unknown variables in $\neg$N
3. **if** $\neg$N is invalid return;
4. insert $\neg$N into the nogood list
5. update nogood list store
6. **check-agent-view**

process-channel($<A_j$, x_j,$pa_j>$) **do**
1. setup relay-channel for A_j

Fig. 5. AAS ok?, nogood and channel processing

- Agents may want to hide private information from others as much as possible.

In searching for a suitable DisCSP algorithm, those above characteristics are the most important criteria for us. Whilst most algorithms such as ADOPT and IDIBT can be extended so that one agent can hold more than one variable, substantial effort is required for this and for handling complex private constraints. In addition, the back-jumping technique in some algorithms require undesirable revelations of agent consumers' addresses to others. Asynchronous Aggregate Search (AAS) [16] is a good candidate since it allows one agent to maintain a set of variables and these variables can be shared. AAS differs from most of existing methods in that it exchanges aggregated consistent values (in contrast to a single value in ABT) of partial solutions. This reduces the number of backtracks. However, private information is revealed more in AAS than in ABT because of the aggregation,. Using trusted servers where critical information of organizations is hosted to prevent privacy loss is not very practical for Web services. In this work, we rate privacy as the most important issue. Also, our goal is to develope a simple but effective algorithm in which agents can exploit special characteristics of the QoS composition problem. As such, we propose an extension of AAS with three following enhancements and modifications:

- Incorporating local CSP solvers into agents to solve complex private constraints.
- Making use of common characteristics of QoS parameters to speed up the solving process.
- Reducing the aggregation in AAS into one value per variable to reduce privacy loss.

Similar to AAS, in our algorithm called constraint satisfaction Algorithm for Distributed Environment (ADE), agents are assigned with priorities so that they can be arranged in order. A composite Web service provider always has higher

priority than the providers of its components. Every agent proceeds with a similar process. Information about the outside world learnt by each agent is stored in an agent view and a nogood list. An agent view is a set of values that the agent believes to be assigned to the variables belonging to the higher priority agents. Agents exchange assignments and nogoods. An assignment has the form $\{\bigwedge_m^{i=1}(x_i{=}a_i),h_j,pa_k\}$ which indicates that the variable x_i is assigned a value a_i. h_j is the message history [16], pa_k is the pseudo-agent address and will be explained later. The assignment is an AAS aggregation of single value. A nogood list holds the assignments of values to the variables during the solving process which cause inconsistency. '*ok?*', or '*nogood'* messages are used as in AAS. The '*ok?*' message is used to inform the lower priority agents of new value assignments. A nogood is used to backtrack the assignment that causes inconsistency between constraints. In Figure 6, we use A^+ to denote the set of agents that are linked to A and have priorities higher than the priority of A. Similarly, A^- is denoted as the set of agents that are linked to A and have priorities are lower than the priority of A. V^+ is the set of variables the agent share with A^+, and V^- is with A^-. As illustrated in Figure 6, when an agent receives an '*ok?*' message, it validates the message and adds the assignment in the message into its local view. It then checks the local view consistency. To handle the complexity of local constraints, we use a local CSP solver inside each agent. In the *check-local-view* procedure, first the agent updates any new assignment. It detects if there is any unknown variable in the assignment. If there is such a variable then the agent asks the message sender to setup a *relay channel* so that it can communicate to the agents who hold this variable. In addition, it generates a pseudo-agent address for each of those agents. These pseudo-agent addresses will be mapped to the actual agent addresses at the sender. In essence, the *relay channel* is equivalent to the *add-link* mechanism in ABT. However it protects an agent from revealing its address to unknown agents by relaying all messages through a known agent. In the *check-local-view*, if the agent view is inconsistent then the agent's local CSP solver is invoked to find new value assignments for this agent's variables. If a solution is found, the agent sends new 'ok?' messages with the assignments to the lower priority agents; otherwise, it sends a *nogood* message back to the *'ok?'* message sender. The detailed description of the extended AAS algorithm and performance analysis is a subject of another paper (under review).

In the context of Web service composition, QoS parameters have particular characteristics that normal constraint variables have not. Agents (service providers) often have had their constraint preference as a monotonic function over a QoS value. In other words, there is a preference operator $\preceq$ defined between any two values $a^{(1)}$ and $a^{(2)}$ in the domain of variable x. If $a^{(1)} \preceq a^{(2)}$ then we say that the agent prefers $a^{(2)}$ to $a^{(1)}$ for x. In addition, this operator can also be defined in the aggregation of assignments as following: $\bigwedge_m^{i=1}(x_i{=}a^1{}_i) \preceq \bigwedge_m^{i=1}(x_i{=}a^2{}_i)$ if $\forall$i=1..m $a^1{}_i \preceq a^2{}_i$. Note that this relation defines only on a subset of the Catersian products of aggregatations. The

check-agent-view do
1. **when** agent-view and current-aggregate are inconsistent
2. V =localCSP.solve(agent-view, nogood-list, local-constraints)
3. **if** V is null
4. backtrack
5. **else**
6. reset current-aggregate
7. **for** $\forall$ a$\in$ V do
8. **if** a is new for $A^+{}_k$
9. append new history and send ok message to $A^+{}_k$
10. current-aggregate = current-aggregate$\cap$a
11. **else**
12. **if** a is needed
13. current-aggregate=current-aggregate$\cap$a

Fig. 6. AAS check-agent-view

preference for A2 in Figure 2, for example, can be that
t(FindAttraction)=1$\bigwedge c$(FindAttraction)=5$\preceq$
t(FindAttraction)=2$\bigwedge$ c(FindAttraction)=4
While the ADE algorithm follows the principles of AAS, the local CSP solver uses a new heuristic for selecting new values. When a new assignment is generated a heuristic criterion could be to choose the value with not less preference (if it is comparable to the previous rejected assignments) to the lower priority agent. This preference is further exploited in our Fuzzy DisCSP algorithm in the next section.

B. FADE algorithm

Our Fuzzy constraint satisfaction Algorithm for Distributed Environment (FADE) algorithm is an extension of ADE for Web service providers with preferences. As pointed out in [8], a FCSP can be modelled as a collection of crisp CSPs at different levels of constraint satisfaction. Using the well-known principle of the resolution identiy [20,21] in fuzzy set theory each fuzzy constraint C_j^f in definition 2 can be decomposed into a union of non-fuzzy (crisp) constraints that are its α-cuts as follows:
$C_j^f(x) = \sum \alpha C_j^{f,\alpha}(x)$
in which $C_j^{f,\alpha}(x)$ is an α-cut of C_j^f defined as
$C^\alpha(x) = \{x \mid \mu_{C_j^f}(x) \geq \alpha, 0 \leq \alpha \leq 1\}$,$\sum$ is the union of $C_j^{f,\alpha}$.
Figure 8 shows an example of four α-cut levels at 0.25, 0.50, 0.75, and 1.0 for fuzzy constraints on three different agents. Following the resolution identity principle, all operations on fuzzy constraints can be performed on the collection of crisp constraints corresponding to the same α-cut levels. A Fuzzy DisCSP in general can be considered as a union of crips DisCSPs at different levels of constraint satisfactions:
DFCSP = $\sum \alpha DisCSP^\alpha$
where any solution of a $DisCSP^\alpha$ has a satisfaction degree greater or equal α. The solution of the Fuzzy DisCSP in definition 2 is to find a solution at the highest α-cut level.

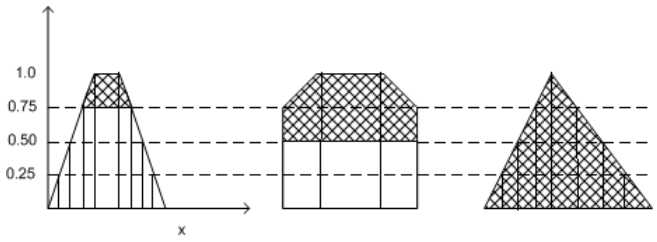

Fig. 7. Searching on different α-cut levels for agents with different satisfaction priorities

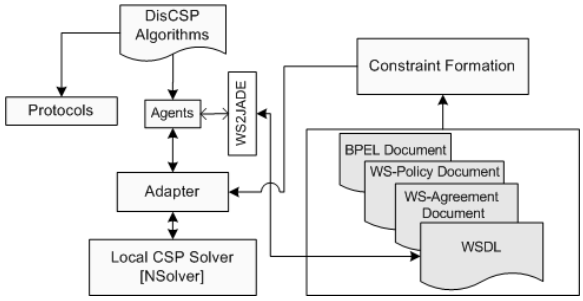

Fig. 8. Different J4WSM system components

This hints us to start searching from the highest α-cut level and move down rather than solving all $DisCSP^{\alpha}$. This is also the principle of our algorithm.

FADE models the Fuzzy DisCSP as a set of crisp DisCSPs at different α-cut levels. It solves a crisp DisCSP by using the ADE algorithm. The solving process starts by having on agents agree on a set of α-cut levels. At each agent, the search begins from the highest level. If a solution in this level can not be found, the search then concedes to the next α-cut level. It is important to know that at the begining every agent start with their highest α-cut levels of 1.0. However during the search, these α-cut levels can become different. Figure 8 illustrates this. Assuming that the satisfaction priorities of agents in the figure decrease from left to right. The figure depicts a search running on the shaded areas: at the α-cut level of 0.75 for the left-hand side agent, 0.5 for the middle agent, and 0.25 for the right-hand side agent. Agents with higher satisfaction priority have higher α-cut level at any instance of time if possible. In more details, the solving process of Fuzzy DisCSP happens in three phases: α-cut level forwarding, DisCSP solving, and level conceding. The second and third phases can be iterative. In the level forwarding phase, every agent forward their α-cut levels of preference to others and together they assemble a global set of α-cut levels. In the DisCSP solving phase, the agents use asynchronous backtracking technique (ADE algorithm) to solve the crisp DisCSP at assigned α-cut levels. The third phase is triggered by the second phase when no solution can be found. Agents which causes no solution, starting from the ones with lowest satisfaction priority, are asked to move to the next α-cut level. The second and third phases are iterated util the time that a solution is found.

In FADE, in addition to the nogood content in a nogood message. The sender also creates the explanation for this nogood and sends along with the nogood message. The explanation is created as follows. An agent during the search for a new solution in *check-local-view* procedure, it takes the union of violated constraints. If the union set is not empty, then all α-cut constraints in this set forms the explanation. Otherwise, the set of α-cut constraints in every recorded nogoods plus the current α-cut constraint make up the explanation. The receiver of the nogood message records this explation in assocation with the nogood and remove any nogood with have the α-cut level greater than the current searching α-cut level. These

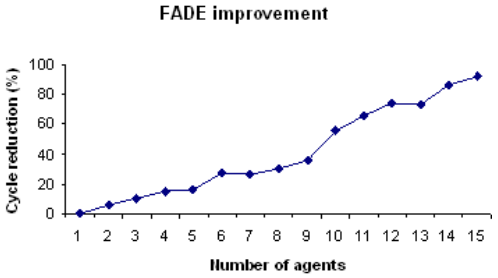

Fig. 9. FADE improvements over Iterative DisCSP algorithm [6] for agents with the same satisfaction priorities

remove possible duplicated searches at different α-cut levels and hence increase the efficiency of the overall search.

VI. Implementation

In this part, we describe our initial agent-based implementation of a framework for DisCSP with our toolkit J4WSM (Jade for Web Services Management) to support our proposal. J4WSM is a re-factor of our previous toolkit called WS2JADE that integrates Jade agents with Web services [14], towards WS management. J4WSM, which consists of a Jade agent platform and other utilities including WS4JADE, runs as a service inside a J2EE container. In this initial version, J4WSM runs under JBoss. J4WSM is similar to Blue-Jade [4] but targets Web Services Management particularly. J4WSM is designed to use services instead of making API calls.

A. FADE algorithm

The main components in J4WSM consists of three main modules: constraint formations, local CSP solver and DisCSP protocols. The constraint formation module is to construct and keep the constraints available at each provider. Constraints in J4WSM are presented in XPath expressions. At this version, constraints are created from composite service structures (in BPEL format) by using the algorithm specified in Section 3. In J4WSM, we do not specify any particular CSP solver engines. Different CSP solver engines can be used as long as there is an adapter which can translate XPath expressions into the CSP solver languages. For our experiment, we use NSolver [15] for the local CSP solver. Whenever a provider needs to solve the local constraints, it invokes NSolver to

find a solution. We have developed an adapter to translate XPath expression into NSolver on the fly. This adapter can be downloaded from [20]. In this version, J4WSM only supports one Fuzzy DisCSP protocol which corresponds to the FADE specified in the previous part. This protocol is developed as an interaction protocol in Jade. Each provider in the DisCSP is represented by a Jade agent. These agents must understand a protocol and agree to use it before the solving process can start. Different components in J4WSM are presented in figure 8. Figure 9 shows the improvement of FADE over the interative DisCSP algorithm in [6]. The graph shows the average improvement of 10 trials for each value in the horizonal axis. The improvement is measured in term of processing cycles which has been poplular used to compare performance between different DisCSP algorithms. For each agent, one cycle consists of reading all incoming messages, invoking its CSP solver to find a solution and sending messages [23]. It can be seen that the reduction of cycle increases with the number of agents and reaches around 90% for 15 agents. The structure of the composition in this experiment is built incrementally by adding one agent each time from the first agent A_1 until a balanced binary tree with depth =4 (i.e there are 15 agents in total) is formed. The experiment is carried out for two parameters: cost and response time. Each agent A_k has a business rule: $c(A_k) \geq \frac{max\{T_0 - t(A_k), 0\}}{T_0} C_T$ in addition to the constraints deduced from the structure of its own composite service: $t(A_k)=t_{own}(A_k)+ \sum_{\forall A_q \in A^+{}_k} t(A_q)$,$c(A_k)=c_{own}(A_k) +\sum_{\forall A_q \in A^+{}_k} c(A_q)$ in which $t(A_k)$and $c(A_k)$are response time and cost of the composite service at A_k, $t_{own}(A_k)$ and $c_{own}(A_k)$are response time and cost introduced by A_k's own service respectively. In the experiment, the domain of the cost variable for an agent with depth=i is $[1..2^{5-i}]$ and the domain of this agent's time response variable is also $[1..2^{5-i}]$. T_0=4ms and C_T=1$. Each provider has a reference function described by a fuzzy membership function with four crisp numbers at the levels of 1.0, 0.75, 0.5 and 0.25. The purpose of this experiment setup is to mimic a real scenario of Web service composition and demonstrate the applicability of DisCSP technique into the QoS composition problem. Figure 10 shows a debug trace for 4 agents with J4WSM in which the agent performative PROPOSE is used for ok? and REJECT-PROPOSAL is for nogood.

VII. Conclusions

This paper proposes a new approach of Fuzzy DisCSP application into the QoS composition problem. An algorithm to construct constraints from the composition topology is proposed. We also develop an enhanced version of AAS for multiple variables with a new heuristic for distributed agents to exploit QoS parameters' characteristics. In addition, the iterative algorithm to solve the QoS composition problems with different Web service providers' preferences - FADE is described. Our future work will concentrate on a framework which allows agents to exchange not only assignments but also possible constraints during the solving process, as a part of J4WSM toolkit. We believe that this is beneficial for the QoS composition problem and useful in developing new Fuzzy DisCSP algorithms for the solving process.

References

[1] V. Agarwal, K. Dasgupta, N. Karnik, A. Kumar, A. Kundu, S. Mittal, and B. Srivastava. A service creation environment based on end to end composition of web services. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 128–137, New York, NY, USA, 2005. ACM Press.

[2] S. Ali, S. Koenig, and M. Tambe. Preprocessing techniques for accelerating the dcop algorithm adopt. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 1041–1048, New York, NY, USA, 2005. ACM Press.

[3] R. Bejar, B. Krishnamachari, C. Gomes, and B. Selman. Distributed constraint satisfaction in a wireless sensor tracking system. In *Workshop on Distributed Constraints, IJCAI*, 2001.

[4] G. M. Cowan, D. and B. B. BlueJade-A service for managing software agents. Hp technical report, HP, Murray Hill, New Jersey, 2002.

[5] X. Gu, K. Nahrstedt, R. Chang, and C. Ward. Qos-assured service composition in managed service overlay networks, 2003.

[6] K. Hirayama and M. Yokoo. An approach to overconstrained distributed constraint satisfaction problems: Distributed hierarchical constraint satisfaction, 2000.

[7] M. C. Jaeger, G. Rojec-Goldmann, and Mühl. QoS aggregation for service composition using workflow patterns. In *Proceedings of the 8th International Enterprise Distributed Object Computing Conference (EDOC 2004)*, pages 149–159, Monterey, California, USA, 2004. IEEE CS Press.

[8] R. Kowalczyk. On negotiation as a distributed fuzzy constraint satisfaction problem. In *Proceedings of the Third International Symposium on Soft Computing for Industry of the World Automation Congress*, pages 631–637, 2000.

[9] Y. Liu, A. H. Ngu, and L. Z. Zeng. Qos computation and policing in dynamic web service selection. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 66–73, New York, NY, USA, 2004. ACM Press.

[10] X. Luo, N. Jennings, N. Shadbolt, H. Leung, and J. Lee. A fuzzy constraint based model for bilateral multi-issue negotiations in semi-competitive environments.

[11] X. Luo, J. H. man Lee, H. fung Leung, and N. R. Jennings. Prioritised fuzzy constraint satisfaction problems: axioms, instantiation and validation. *Fuzzy Sets Syst.*, 136(2):151–188, 2003.

[12] D. A. Menasce. Composing web services: A qos view. *IEEE Internet Computing*, 8(6):88–90, 2004.

[13] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees.

[14] X. T. Nguyen. Demonstration of ws2jade. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 135–136, New York, NY, USA, 2005. ACM Press.

[15] *NSolver home page*. www.cs.cityu.edu.hk/ hwchun/nsolver/, 2005.

[16] M. C. Silaghi and B. Faltings. Asynchronous aggregation and consistency in distributed constraint satisfaction. In *Artificial Intelligence Journal Vol.161*, pages 25–53, New York, NY, USA, 2005. ACM Press.

[17] B. Srivastava and J. Koehler. Web service composition - current solutions and open problems. In *ICAPS 2003 Workshop on Planning for Web Services*, 2003.

[18] W. M. P. van der Aalst. Don't go with the flow: web services composition standards exposed. *IEEE Intelligent Systems*, 18(1):72–76, 2003.

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

[20] *XPath Adapter for NSolver*. www.it.swin.edu.au/centres/ciamas/tiki-index.php?page=xpath2nsolver, 2005.

[21] *Web Services Choreography Description Language Version 1.0*. http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/, 2006.

[22] M. Yokoo. Asynchronous weak-commitment search for solving distributed constraint satisfaction problems. In *Proc. 1st Intrnat. Conf. on Const. Progr.*, pages 88–102, Cassis, France, 1995.

[23] M. Yokoo and K. Hirayama. Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems*, 3(2):185–207, 2000.