A Constraint-Based Approach for Developing Consistent Contracts in Composite Services

Basem Suleiman [1,2]
[1]*NICTA*
Australian Technology Park (ATP)
Sydney, Australia
[2]*School of Computer Science and Engineering*
University of New South Wales
Sydney, Australia
Basem.Suleiman@nicta.com.au

Fuyuki Ishikawa
GRACE Center
National Institute of Informatics (NII)
Tokyo, Japan
f-ishikawa@nii.ac.jp

***Abstract*—** *A key problem that challenges the designers of service-oriented systems is ensuring the consistency of composite Web service contracts based on their parameters. This paper utilizes constraint satisfaction approach to examine the problem at design time and by focusing on quality of service (QoS) contract parameters. It proposes a generic framework to formalize service contract composition as a constraint satisfaction problem (CSP). It also introduces an initial tool design for automating composite contract consistency checking and adaptation based on QoS parameters. The tool aims at supporting Web service orchestrators to specify appropriate contract parameter values and adapt them so that consistency of composite contracts is increased to some extent. Further, it enables them to analyze and reason about violation percentages during contract negotiation phase. The benefits of the proposed CSP framework and the tool design have been illustrated through a Stock Manager Web service composition scenario.*

Keywords- composite contracts; Web service composition; consistency checking; quality of service parameters; constraint satisfaction;.

I. Introduction

Due to its proved benefits, service-oriented computing (SOC) has become the most dominating engineering paradigm for software systems. In SOC, software components provide services to other applications through published and discoverable interfaces [1], and hence they are called Web services. The process of orchestrating different Web services into a new service is a key concept in SOC which is called service composition. Service composition may depend on different criteria including service properties (e.g., functional and quality of service properties). Such properties are usually specified in the service contract as guarantees and conditions of service behavior. Flexibility in service composition depends on the extent to which service providers allow for changes in their contracts, i.e., service properties.

Contract engineering and management in Web service orchestration are among the most essential activities which challenge Web service designers because they play a central role in constructing consistent and reliable composite services. Web service contracts define commitments and guarantees that each service provider should respect for all service consumers. Web service level agreement (WSLA) [2] and The Web Service policy Framework (WS-Policy) [3] are among the most adopted industry standards for contract specifications. WSLA is an IBM standard language that allows specifying agreed-upon guarantees for IT-level service parameters such as availability and response time. WS-Policy is another standard for specifying Web service capabilities and constraints. Composing services require precise selection and specification of contract parameters' values that ensure minimum violation of the composite contracts. Unfortunately, WSLA and WS-Policy cannot help in deciding the most appropriate parameters values of composed contracts- they focus on providing standard language constructs to facilitate contract specification. Furthermore, it is almost impossible for service providers to provide 100% guarantees for their contract parameters. Therefore, the specification of parameters values of composed contracts at design time and their monitoring at run-time are crucial tasks for the service orchestrators. They need techniques to aid them predicting possible contract violation and consistency when composing Web services and during its execution. This research focuses on the former issue, i.e., helping service designers to specify appropriate values for QoS contract parameters that ensure the development of consistent composite services.

In contract composition, QoS parameters and their values are usually specified based on QoS parameters values of composed services. Service providers normally need to provide QoS parameter values that waive their responsibility of any violation or failure. On the other hand, composite services seek specifying QoS parameters values that ensure best and competitive guarantees for its consumers. These needs often lead to an on-going negotiation between service providers and the orchestrator of the composite service. During this process, continuous changes to the QoS parameters values take place and require consistency checking of the composite contract parameters values. For example, the service orchestrator needs to know the percentage of violation that could result from changing one or more of its QoS parameters values. They also want to ensure minimum violation or failure that could result from changing one or more QoS parameter values of one or more of the provided services.

Most of the composite contract studies define QoS contract parameters as hard limits. For instance, response time need to be less than an exact value and number of allowed inquiries must not exceed a certain number per second. Although this approach helps service providers to provide more reliable guarantees, it challenges service orchestrators with specifying impractical QoS parameters values in the composite contract. For example, response time needs to be less than the summation of all response times of all provided services (while in this research we consider such simple formula, but in real situations the response time would be calculated through a complex formula that considers different variables.) While such summation results in a high response time, the service orchestrator in practice is required to provide a competitive response time guarantee to its consumers, i.e., as minimum as possible. Such requirements usually challenge the service orchestrators with the need to specify, vary and analyze several possible contract parameters values (both provided and composite parameters values) so that the possibilities that would lead to inconsistent composite contract can be reduced. This research aims at supporting the service orchestrators in achieving such tasks during composite contract construction and negotiation by:

1) Providing a generic formalization framework of contract composition problem as a constraint satisfaction problem (CSP) during design-time.

2) Designing a support tool that enable design-time consistency checking of composite contracts based on provided QoS parameters values during contract development and negotiation process. It aims at facilitating flexible negotiation and re-design of composite contracts based on QoS contract parameters to ensure reduction of possible contract violations. The objectives are intended to aid Web service designers in negotiating with service providers about contract parameters values and constructing consistent composite contracts through exploring and analysing appropriate combinations of QoS parameters values of all involved service contracts. By consistent we mean appropriate values specification to the parameters of the composite contract that reduce possible contract violation during service execution.

Section II motivates to our research objectives through a practical Scenario. Section III describes the generic framework for modeling composite contract problem as a CSP. Section IV applies the proposed framework to the scenario and its use as tool for consistency checking is also discussed. Solving techniques for the CSPs are discussed in Section V. The discussion of the proposed approach is introduced in Section VI. Finally, related work and conclusions are introduced in Section VII and VIII respectively.

II. *Motivating Scenario*

To illustrate the objectives of our research, we present how a Web service composition example for financial stock trading could benefit from our approach for composite contract formalization using CSP to analyze and ensure consistency of QoS contract parameters values. In the following Sections, we discuss how these objectives are achieved and demonstrate how the stock trading Web service composition to some extend validate the feasibility and usefulness of our proposed approach.

Stock Manager is a composite service that provides stock information for its consumers to help them deciding whether and when to buy and/or sell financial stocks. As shown in Fig. 1, it is composed from several specialized Web services namely, Stock Quotes, Market Financial Trends, Expert Advice and Currency Exchange. The properties of these services are documented in the form of service contracts. Fig. 1 shows only essential services' parameters in the context of this research (other contract information is not shown in the example.) Examples of such guarantees include Stock Quotes promises response time to be less than 70 ms, Market Financial Trends allows throughput (i.e., maximum number of allowed queries per hour) to be 160 queries, the validity of the information provided by Expert Advice is up to 15 minutes, the call cost of Currency Exchange service is between \$0-\$10 depending on the request time and other contextual variables. The complete details of contract parameters are shown in Fig.1. The number of contract parameters of the provided services and their types could vary from one case to another. They also may depend on many other contextual aspects such as measurement methods, software and hardware infrastructure specifications and internal control structure of a Web service, to name a few. As we narrow down the scope of our research to concrete services, such aspects are not dealt with in this study.

Specifying optimal and competitive values of Stock Manager's contract parameters is a challenging task for the service orchestrator. Usually there are several dependencies between contract parameters of the provided services and Stock Manager. For instance, the relationship between Stock Manager's response time and provided services' response times could be represented as follows:

$$SM.RT \leq \partial + SQ.RT + MFT.RT + EA.RT + CE.RT$$

Where ∂ is the Stock Manager overhead. In practice, the formula would be more complex than this one, but for simplicity, we assume that response times (and other QoS parameters) of provided service already considered other aspects and the provided values are the ultimate ones. The problem becomes more challenging when the number of involved services in the composition becomes large and various changes occur in one or more different QoS parameters such as response time and information validity. Contract violation could result from any QoS parameter type (e.g., throughput or information validity) at any particular point of time during composite service execution.

To specify a competitive response time value for Stock Manager, the service orchestrator needs to ensure percentage of inconsistent and consistent cases that would result from various value combinations of provided services' response

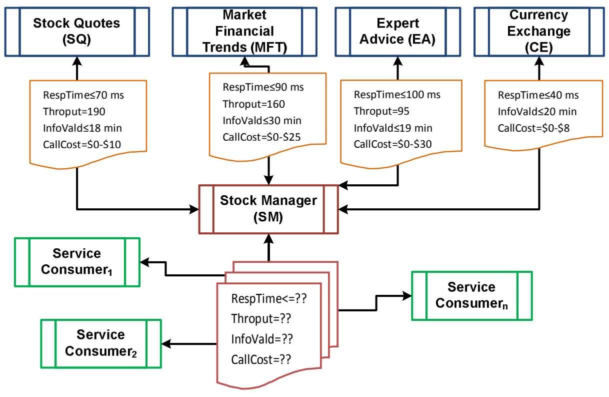

Figure 1. Stock Manager composite service and contract parameters

times (or any other QoS parameters.) Similarly, if one or more response time of the provided services changes, the service orchestrator wants to find out the impact on the number of consistent/inconsistent cases and therefore their percentages. Such activities are crucial for service orchestrators as they need to figure. out percentages of possible consistent/inconsistent cases of their composite contract so they can adapt them or describe them in a more precise way. Even service orchestrators would need providing different versions of composite service contract based on consumer type. This requires varying the values of QoS contract parameters values and checking the percentage of violation accordingly. For example, they could provide QoS parameter values that have low violence rate for their crucial service consumers.

These challenges are among the motivating drivers for our adoption of a constrained-based approach for developing consistent contracts in service composition. The next Section introduces our proposed generic framework for formalizing the composite contract problem as a CSP.

III. *Formalization of Composite Contract as Constraint Satisfaction Problem*

The constraint satisfaction (CS) approach has been recently used as framework for modeling and solving complex problems, specifically for combinatorial ones [4]. It has been successfully applied to real-world problems in various areas such as planning and scheduling. Problems that are modeled using constraint satisfaction approach are called constraint satisfaction problems (CSPs). CSP is a problem that consists of different variables where relations between these variables are stated as constraints.

CSP is formally defined as triple (V, D, C) where V is a finite set of variables, D is a finite set of possible values that each variable in V can take (i.e., its domain) and C is a set of constraints that restrict the values that each variable can take at the same time. A solution to a CSP is a value assignment to each variable from its domain so that all constraints are satisfied simultaneously. Based on this definition, there could be:

a) *One solution*: any variable-value ordering that satisfies all constraints without any preference or selection criteria (e.g., the first solution that the algorithm could found.)

b) *A number of solutions*: all possible variable-value orderings that each of which satisfies all constraints at the same time. In this case, solutions can be sorted based on an objective function which is not in the scope of our study.

Our formalization of composite contract problem (*CCP*) is based on the above CSP definition. A *CCP* can be defined as ***(SP, CV, CVD, CC)*** where:

- ***SP = {sp1, sp2 … sp$_n$}***, *is a set of Web service providers from which composite contract is developed.* We assume that these service providers are atomic, i.e., they are not composed of other services.
- ***CV = {CV1, CV2… CV$_n$},*** *is the set of contract parameters (i.e., variables) of Web service providers, where:*

  $CV_1 = \{ sp_1.cv_1, sp_1.cv_2 \ldots sp_1.cv_i\}$
  $CV_2 = \{ sp_2.cv_1, sp_2.cv_2 \ldots sp_2.cv_j\}$
  …
  $CV_n = \{ sp_n.cv_1, sp_n.cv_2 \ldots sp_n.cv_k\}$

Similarly, these contract parameters are atomic not composite ones. Each service provider may have different number of contract parameters. Web service contracts may contain different kinds of parameters or variables such as quality of service (QoS) parameters, utility parameters and resource parameters. The focus of this study is set on the QoS variables such as response time, throughput and availability. To the best of our knowledge, most composition studies in the literature focus on such QoS attributes due to their direct influence on the overall service quality. Some other contract parameters which seem not to have clear classification such as information validity are within the scope of our study. Nevertheless, our framework can be used for modeling any kind of contract parameters, but further aspects need to be considered.

- ***CVD = {{CVD1}, {CVD2}… {CVDn}},*** *is the set of domain sets over which each contract variable ($sp_n.cv_k$) ranges, where each domain CVD_n consists of a set of ranges depending on the number of QoS parameters.* This can be represented as follows:

  $CVD_1 = \{ sp_1.cv_1d_1, sp_1.cv_2d_2 \ldots sp_1.cv_id_j\},$
  $CVD_2 = \{ sp_2.cv_1d_1, sp_2.cv_2d_2 \ldots sp_2.cv_jd_j\},$
  …
  $CVD_n = \{ sp_n.cv_1d_1, sp_n.cv_2d_2 \ldots sp_n.cv_kd_k\}$

The domains can be of integer, real or Boolean types. Domains could vary from contract variable to another.

- ***CC = {cc$_1$, cc$_2$ … cc$_m$}***, *a set of constraints that represent relationships between contract parameters and restrict the values that each attribute can take at the same time.* The relationships could be between contract variables of the same or different types**.**

Solving composite contract problem is achieved by finding all possible contract parameters value combinations,

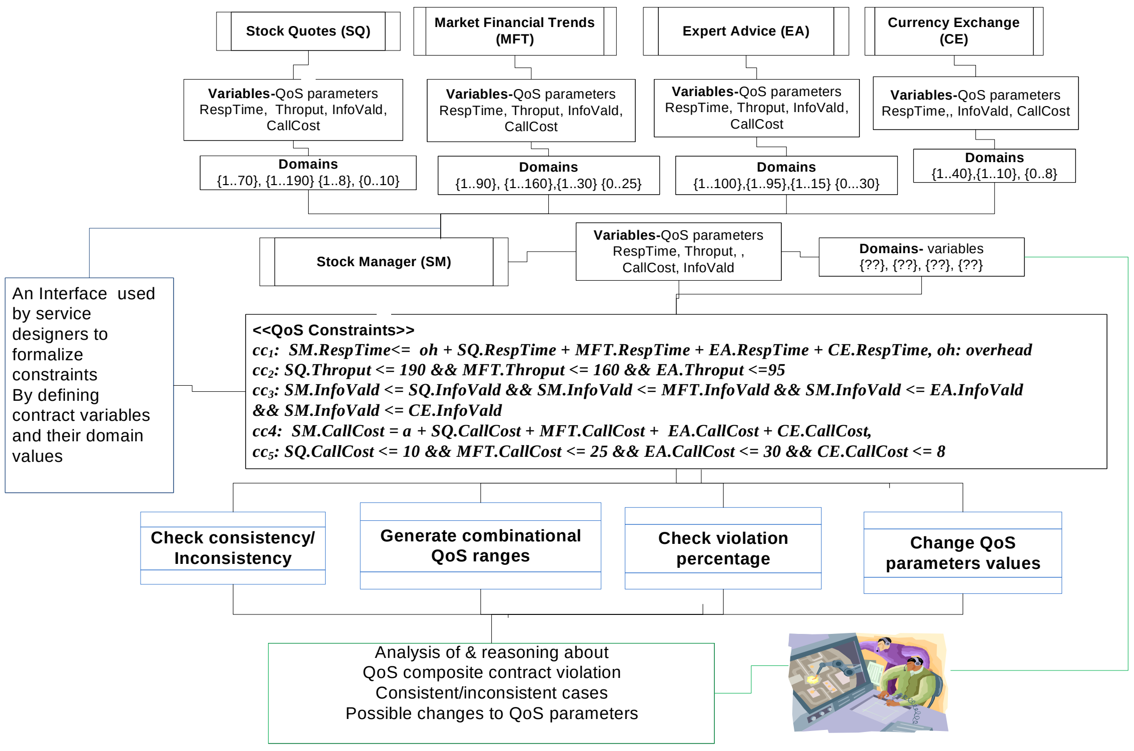
Stock Quotes (SQ)
Market Financial Trends (MFT)
Expert Advice (EA)
Currency Exchange (CE)
Variables-QoS parameters RespTime, Throput, InfoVald, CallCost
Variables-QoS parameters RespTime, Throput, InfoVald, CallCost
Variables-QoS parameters RespTime, Throput, InfoVald, CallCost
Variables-QoS parameters RespTime,, InfoVald, CallCost
Domains {1..70}, {1..190} {1..8}, {0..10}
Domains {1..90}, {1..160},{1..30} {0..25}
Domains {1..100},{1..95},{1..15} {0...30}
Domains {1..40},{1..10}, {0..8}
Stock Manager (SM)
Variables-QoS parameters RespTime, Throput, , CallCost, InfoVald
Domains- variables {??}, {??}, {??}, {??}
An Interface used by service designers to formalize constraints By defining contract variables and their domain values
<<QoS Constraints>>
cc1: SM.RespTime<= oh + SQ.RespTime + MFT.RespTime + EA.RespTime + CE.RespTime, oh: overhead
cc2: SQ.Throput <= 190 && MFT.Throput <= 160 && EA.Throput <=95
cc3: SM.InfoVald <= SQ.InfoVald && SM.InfoVald <= MFT.InfoVald && SM.InfoVald <= EA.InfoVald && SM.InfoVald <= CE.InfoVald
cc4: SM.CallCost = a + SQ.CallCost + MFT.CallCost + EA.CallCost + CE.CallCost,
cc5: SQ.CallCost <= 10 && MFT.CallCost <= 25 && EA.CallCost <= 30 && CE.CallCost <= 8
Check consistency/ Inconsistency
Generate combinational QoS ranges
Check violation percentage
Change QoS parameters values
Analysis of & reasoning about QoS composite contract violation Consistent/inconsistent cases Possible changes to QoS parameters

Figure 2. Application of Stock Manager scenario and basic design of the tool support for consistency composition contract checking

from their domains, such that all constraints are satisfied. In fact, the solution could be represented as a subset of the Cartesian product of all variable domains, i.e., $CCPS \in CVD_1 \times CVD_2 \times \ldots \times CVD_k$. In this research, the focus set on deriving combination of parameters values that ensure consistency of a composite contract to a certain percentage. Finding the optimized solution is out of this research scope.

IV. *Developing Consistent Composite Contracts*

In this Section we introduce an approach that utilizes our CSP framework (presented in the previous Section) to help the service orchestrator to analyze, check and change values of contract parameters when developing composite contracts. The approach is implicitly depicted in Fig. 2 which is explained later. It is summarized in the following steps:

I. *Identify the selected service providers that are involved in the service composition (**SP_n**) and get their service contracts.*
II. *For each service provider (**SP_n**), identify all contract QoS parameters (**$SP_n.CV_k$**) and define them as variables.*
III. *For each defined QoS variable (**$SP_n.CV_k$**), identify its domain (**$SP_n.CV_kD_k$**) through which it may range.*
IV. *Derive the constraints (**CC_m**) of the composite contract using the identified variables and domains. At this stage this derivation needs to be developed manually by service orchestrators. It requires finding possible relationships among contract variables. Relationships could exist between similar and/or different contract variables of different service providers. Past experiences, guidelines and best practices can be utilized to build such constraints. In sub-section B we discuss the interface design to aid the designers in constructing such constraints.*
V. *Specify initial values for the composite contract QoS parameters based on historical information.*
VI. *Run the CSP solver algorithm(s) to test violation percentage, and/or consistent/inconsistent cases, which would result from the value combinations of the identified QoS parameters. Section V discusses solving CSP algorithms.*
VII. *Analyze the generated cases and violation percentages and adapt the composite contract parameter values accordingly. Negotiate with service providers about their contract parameter values until an agreed state is reached.*

A. *Stock Manager Contract Development*

Based on the above steps and the proposed framework, we now apply the approach to the motivating scenario. The Stock Manager composition can be defined as follows: ***SMC = (SP, CV, CVD, CC)***, where the involved services in the composition are:

SP = {SQ, MFT, EA, CE, SM}

The contract QoS parameter variables (***CV***) of each service provider are:

CV = {SQ_CV, MFT_CV, CE_CV, SM_CV, SM_CV}

SQ_CV = {SQ.RespTime, SQ.Throput, SQ.InfoVald, SQ.CallCost}
MFT_CV = {MFT.RespTime, MFT.Throput, MFT.InfoVald, MFT.CallCost}
EA_CV = {EA.RespTime, EA.Throput, EA.InfoVald, EA.CallCost}
CE_CV = {CE.RespTime, CE.InfoVald, CE.CallCost}
SM_CV = {SM.RespTime, SM.Throput, SM.InfoVald, SM.CallCost}

The Stock Manager has considered as a service provider because it will provide its services to other parties (i.e., service consumers as shown in Fig. 1). However, it still consumes services provided by the other four services and thus it has relationships with them. Further, we need to distinguish its contract parameters from other services' parameters. The domains ***CVD*** of the defined variables are:

CVD = {SQ_CVD, MFT_CVD, EA_CVD, CE_CVD, SM_CVD}

SQ_CVD = {1…70, 190, 8, 0…10}
MFT_CVD = {1…90, 160, 30, 0…25}
EA_CVD = {1…100, 95, 15, 0…30}
CE_CVD = {1…40, 10, 0…8}
SM_CVD = {??,??,??,??}

Note that the domains of Stock Manager (***SM_CVD***) need to be derived and changed from ***SQ_CVD - CE_CVD*** and based on the constraints defined below. In other words, we need to find the values of all Stock Manager' QoS parameters that satisfy all the below constraints simultaneously. Furthermore, the service orchestrator may need to know the percentage of violation of Stock Manager contract when one or more of its or service providers' QoS parameters are changed during the negotiation phase. The constraints on this contract composition are:

CC = {cc_1, cc_2, cc_3, cc_4, cc_5}
cc_1: SM.RespTime ≤ ∂ + SQ.RespTime + MFT.RespTime + EA.RespTime + CE.RespTime, where ∂ is the internal overhead of Stock Manager service.
cc_2: SQ.Throput ≤ 190 && MFT.Throput ≤ 160 && EA.Throput ≤95
cc_3: SM.InfoVald ≤ SQ.InfoVald && SM.InfoVald ≤ MFT.InfoVald && SM.InfoVald ≤ EA.InfoVald && SM.InfoVald ≤ CE.InfoVald
cc_4: SM.CallCost ≥ α + SQ.CallCost + MFT.CallCost + EA.CallCost + CE.CallCost, where *α is the sales commission*
cc_5: SQ.CallCost ≤ 10 && MFT.CallCost ≤ 25 && EA.CallCost ≤ 30 && CE.CallCost ≤ 8

Fig. 2 (the upper half including the constraints) depicts the formalization of the Stock Manager composite contract scenario and shows the relationship between contract parameters of the provided services and the composite one. The second part of the Figure (the lower part) corresponds to the tool design which is discussed in details in the next sub-section.

Suppose that the Stock Manager orchestrator decides to allocate the following initial values (domains) for the Stock Manager contract parameters:

SM_CV= {SM.RespTime, SM.Throput, SM.InfoVald, SM.CallCost}
SM_CVD = {1...160, 1...150, 1...16, 50…75*}*

Now s/he needs to analyze the percentage of contract violation that would occur as a result of value combinations of QoS contract parameter values. By entering all the scenario contracts' parameter variables, their values and constraints to a solver algorithm that checks violation of constraints (discussed in Section V) we can get combination of QoS contract parameter values that that lead to consistent or inconsistent composite contract (based on the defined constraints.) Fig. 3 and 4 show samples of such consistent and inconsistent cases respectively. The algorithm will iterate through all possible value assignments to the problem variables and check satisfaction of all constraints simultaneously.

As shown in Fig. 3 (first set of ranges), although all response times of service providers are respected, but their total violates the cc1 constraint. In case 2, the Information validity values of SQ, EA and CE contracts are less than what SM promises. There are many other value combinations that lead to such contract violation. Similarly, Fig. 4 shows some contract parameter value combinations that ensure satisfaction of all composite contract constraints at all times. The designers need not to worry about finding such cases as the tool supposes to do so. Instead, they can analyze and reason about most critical cases and consider them for negotiation with service providers and re-design the composite contract based on assigning new values (to the composite contract parameters) which lead to higher consistent states. Based on these consistent/inconsistent cases they can also find percentage of contract violation of particular composite contract parameter values. Furthermore, they can keep tuning these values and/or negotiating with the service providers about provided contract parameter values until they reach certain percentage of consistency or violation which can be accepted by consumers of the composite service.

Stock Manager contract violation- value combinations
…….
{{**40**,52,18,5},{**67**,15,23,13},{**55**,10,18, 18},{**20**,20,2}}
{{31,52,**10**,5},{24,15,27,13},{41,10,**15**, 18},{18,**10**,2}}
{{**53**,190,**9**,5},{**72**,160,26,13},{**63**,95,**12**,18},{**32**,**11**,2}}
……

Figure 3. Samples of SM contract violation cases

Stock Manager consistent contract- value combinations
…….
{{21,92,18,5},{24,86,23,13},{49,10,18, 18},{14,20,2}}
{{35,87,16,5},{15,87,25,13},{42,87,17, 15},{14,15,3}}
…..

Figure 4. Samples of SM contract consistent cases

As a result, different composite contract versions can be generated and agreed upon with different service consumers.

B. *Tool Support for Ccomposite Ccontract Cconsistency Checking*

In this Section we discuss the tool design to support the automation of composite contract consistency checking. As shown in Fig. 2 (the upper part), the designers can use the provided contract parameters and their domains to construct appropriate constraints between them and composite contract parameters (this has been discussed in sub-section A.) Part of our plans for a prototype implementation is the design of an interface that helps the designers to easily and quickly build composite constraints from several service contracts' parameters. The interface aims at facilitating the designer's task by enabling them to select contract parameters from a predefined library and to specify their domain values. It also provides essential mathematical and logical operators that are required to constructing constraints. Such interface would be more useful when it is integrated with WSLA framework.

As shown in Fig. 2 (the lower part), the input to the tool is a set of contract parameters, their domains and the relationship between them, i.e., the constraints. The output depends on the selected functionality. We show some examples of consistent/inconsistent cases generated based on the Stock Manager scenario (see Fig. 3 and 4.) The "*Generate combinational QoS ranges*" functionality enables automatic generation of the domains of the composite contract parameters using the involved service contracts. This would consider different factors such as balancing different parameter values so that some parameters are increased and others are reduced. For instance, finding contract domains that minimize the composite contract response time and maximize the information validity and at the same time have the minimum contract violation percentage. The "*Check violation percentage*" feature will allow the service orchestrator to know the percentage of contract violation that would result during service execution and due to combination of contract parameters values. The orchestrator can then adapt QoS contract parameter values until particular violation percentage is reached (e.g., the average). Further, this can help them in generating different contract versions according to the service consumer type or needs. For example, if the service consumer is a very important client to the composite service, then the service orchestrator needs to find better contract parameter values/ranges that minimize the violation percentage than for a normal service consumer.

In addition to these functionalities, the tool will also provide "*Change QoS parameter values*" to adapt contract parameter values according to certain conditions such as reducing the violation percentage to a certain value. These functionalities are aimed at support the orchestrator's tasks during contract construction and negotiation phase at design. Their output can help the orchestrators to analyze various contract cases and reason about them in a timely and productive manner. Furthermore, it helps the orchestrator to think about witting exceptional handling for critical contract violation cases that cannot be handled with the specified domains.

V. Solver Algorithms for the Composite Contract Consistency Problem

There are several solver algorithms for solving CSP in different domains (see [5] for a list of some constraint solvers). Such solvers are based on systematic search algorithms and Artificial intelligence techniques. Generate-and-test [4] is one of the well-known algorithms which generates all possible value combinations and then test whether or not they satisfy all constraints. Such algorithms can be utilized for generating all possible consistent/inconsistent contract value combinations. In our tool investigation, we conduct some experiments using ZDC-Rostering tool [6], an application that enable modeling and solving CSPs with a focus on Scheduling and planning problems. It provides different solving techniques such as generalized Forward Checking solver, Linear Programming solver and local search solvers and Genetic Algorithms. We experienced efficiency and performance problems with these algorithms. For example, it took the application long time to find all possible value combinations that lead to consistent combinations of contract parameters (based on the provided constraints). Such problems make the approach to somewhat impractical. Therefore, we research for ways on how to improve efficiency of CSP solving algorithms.

Backtracking [4] tries to gradually extend partial consistent value combinations toward a complete one by recurrently selecting a variable value. Obviously, this technique is useful for finding consistent cases. However, late detection of inconsistent cases is a disadvantage of this approach. To explore inconsistent states earlier and reduce search space, different techniques such as node-consistency, arc-consistency and path consistency which are based on constraint graphs [4] could be also used. Forward checking, look-ahead and look-back techniques were resulted from integrating consistency techniques and search algorithms. Full details of all these algorithms and techniques can be found at [4].

To this end, we believe the feasibility of implementing efficient tool (as discussed in the previous Section) is possible. Having said this, the utilization of available algorithms (discussed at [4]) (with some customization) will help achieving the goals of our proposed tool. As there are different functionalities (discussed in sub-section B), there will be a need for implementing different solvers and algorithms to achieve these functionalities. This requires more empirical studies that include, but not limited to, complexity analysis and performance evaluation of existing algorithms, solution's optimization, Implementation and technical details of the proposed model and algorithms. In fact, the prototype implementations of such algorithms and their performance analysis and evaluation requires further individual research studies that are focused on solving and optimizing consistency checking of QoS contract parameters in composite services. These are main parts of our future work.

VI. DISCUSSION

The model presented in Section III aims at providing generic framework for modeling composite contract parameters and their relationships as a CSP. Variables and constraints need not to be represented in any special notations which make them easy to be constructed and understood by service designers. Further, they correspond directly to the real problem entities making them closer to the original problem. Constraints can be solved without the need to be translated into other simplified formulas. In addition, their construction would not be a complicated task as constraints need to be derived from existing contract parameters' values and by using basic mathematical and logical operators. The existence of wide range of algorithmic search and optimization techniques for solving CSP make automated tools development for consistency checking more feasible.

Those variables and constraints are key input for the tool design (presented in sub-section B.) The exploration of consistent/inconsistent cases requires exhaustive searching techniques that generate and test various states. Even changing or tuning contract attributes' values require re-checking of contract consistency in a timely manner. It is almost impossible for designers to achieve such tasks without the tool support. In addition, the tool can facilitate analysis and reasoning tasks. For instance, it can generate percentages of contract possible violation cases that could be caused by each contract parameters or group of them. Accordingly, the designers can then decide on priority of adapting or changing contract parameter values with the highest influence on the composite contract (e.g., response times). Generating inconsistent case of contract values also has its benefits. They can be used as a basis for developing proactive strategies during service orchestration that handle most critical ones through exceptions.

Another benefit of the contract consistency checking tool is to support contract versioning. For composite service it is sometimes essential to develop different contract instances with different parameter values to meet various types of consumer requirements or importance.

TABLE I. CONSISTENCY PATTERNS OF VALUE COMBINATIONS

Attribute values/contract satisfaction	**Composite contract**
Consistency Patterns	
All contract parameter values satisfy their own contract guarantees	Not violated
One or more attribute value violate their contract agreement	Not violated
Inconsistency Patterns	
One or more attribute value violate their contract agreement	Violated
All attribute values satisfy their own contract guarantees	Violated

To achieve this, the designers need to specify the most appropriate values for their contract parameters and their violation percentages Thus, they need to generate various consistent value combinations that meet expectations of their consumers and based on which they precisely create various contract versions.

We assumed that service providers are concrete to narrow down the scope of our study. In reality, provided services may be composed from other Web services. Furthermore, there are more complicated compositions, other than sequence of services, which involve control structures such as conditions and loops. Currently, such structures can be taken into consideration when service designers construct constraints that determine weight of each contract parameter. For example, they can specify high weight on parameters that denote QoS of providers which are invoked many times in a loop execution. While these issues add a layer of complexity that challenges our proposed model, we believe that some existing researches in the literature (e.g., [7]) proposed approaches on how to compute valuations of Web service compositions. Such approaches would support our assumption and help on focusing on other issues.

We found out from the precise analysis of some output samples that although all contract parameter values would satisfy their own contract guarantees, but there are some cases where the composite contract could be breached. Thus, satisfaction of service providers contract attributes would not always lead to consistent composite one. Table 1 shows all possible pattern categories that would result from value combinations.

VII. RELATED WORK

Through examining the literature, we classify the related studies into two main streams namely, formal modeling and specification of consistent contract composition [8, 9] and constrained-based approaches for Web service composition, orchestration and their monitoring [10, 11, 12]. While Ishikawa et al. used event calculus to specify constraints on composite contracts and reason about their consistency, Lamparter et al. built their formal modeling of contracts on DOLCE (Descriptive Ontology for Linguistics and Cognitive Engineering). Our goal is not another formal modeling technique for contract composition, but simple representation of contract parameters and their relationships in order to be used as input for exploring consistent and inconsistent cases in service composition. The designers need not to learn how to write event calculus predicates or ontology axioms to represent and reason about the dependencies between contract parameters. Variables in CSP correspond directly to the real contract parameters (e.g., QoS) making CSP representation closer to the original problem. Furthermore, constraints can be solved without the need to be translated into other simplified formulas. This makes the problem formulation and the problem and the solution easy to understand by both humans and solvers.

Although the constraint-based approaches [10, 11, 12] for dynamic service selection and composition used similar approach (i.e., CSP), but our research objectives differ from these researches. Specifically, our research aims at providing support to service orchestrators during design-time to reason about contract consistency and make decisions for better service orchestration. In contrast, Channa et al. focus on dynamic composition CSP optimization approach based on various aspects such as cost, QoS and process constraints. The agent-based technique [11] utilizes fuzzy distributed CSP to model and solve QoS –based composite contract constraints. Our approach does not consider satisfying all providers' local constraints as it is central to the composite service only, i.e., the orchestrator. The constrained-based service composition in [12] is basically based on the functional requirements of the composite service. In other words, the optimal service selection and execution at run-time depends on user selections and preferences. Our approach focuses on the most common criteria (e.g., contract parameters including QoS) during service design that would cause contract violation and lead to losses.

While Rosario et al. proposed a comprehensive probabilistic approach [13] to soften QoS parameters in service composition, our research deals with QoS parameters in the form of hard bounds. Their tool TOrQuE (Tool for Orchestration simulation and Quality of service Evaluation) enables constructing probabilistic contracts, their composition and monitoring for Web service orchestration. According to our best knowledge, hard constraints are the most commonly used technique for contract agreements and service composition in research and practice.

VIII. Conclusion

This study has introduced a generic framework for modeling composite contracts as a CSP and based on QoS parameters. It has illustrated solving contract composition problem, i.e., specification and adaptation of QoS contract parameters' values of the composite service. The proposed CSP framework and solving approach provide a theoretical foundation for modeling and solving constraints on composite contract so that they facilitate composite contract development. Based on this foundation, we also discussed an initial design for tool support to automate contract consistency checking and analysis. Furthermore, we show how the tool could allow varying QoS contract parameter values and check possible percentage violation that could result during composite service execution. Time saving is one of the obvious benefits for service orchestrator. In addition, our approach will enable developing more reliable and consistent composite contracts once it is implemented.

The implementation of the proposed framework and tool are among the most important items for our future work. Considering other contract parameters such as utility, grid and context parameters is also another essential future work item. In addition, modeling and solving more complicated compositions that involve conditions and loops need to be investigated.

Acknowledgment

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

[1] M. P. Singh and M. N. Huhns, Service-Oriented Computing: semantics, Processes, Agents. John Wiley and Sons, 2005.

[2] IBM, "Web Service Level Agreement (WSLA)",online: http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf, accessed: January 13, 2009.

[3] W3C, Web Service policy 1.5-Framework, online: http://www.w3.org/TR/ws-policy/, April 7, 2009

[4] R. Bartak, On-line Guide to Constraint Programming, available online: http://www.constraintsolving.com/Favlinks.html, access date: January 12, 2009.

[5] Constraint Libraries, online: http://www.constraintsolving.com/ConstraintLibrary.html, access date: February 15, 2009.

[6] E. Tsang, et al., "ZDC-Rostering: A Personnel Scheduling System Based On Constraint Programming", TR 406, 2004, accessed online: http://dces.essex.ac.uk/CSP/papers/TFMBWS-ZdcRostering-TR406_2004.pdf, date accessed: February 5, 2009.

[7] J. Pathak et al., "Modeling Web Service Composition using Symbolic Transition Systems", American Association for Artificial Intelligence, online: http://www.aaai.org/Papers/Workshops/2006/WS-06-01/WS06-01-006.pdf, accessed: April 15, 2009

[8] F. Ishikawa, N. Yoshioka, and S. Honiden, "Developing Consistent Contractual Policies in Service Composition", Asia-Pacific Service Computing Conference, IEEE CS, Tsukuba Science City, Japan, pp. 527-534, December. 2007,

[9] S. Lamparter, S. Lunckner, and S. Mutschler, "Formal Specification of Web Service Contracts for Automated Contracting and Monitoring", 40th Hawaii Int. Conf. on System Sciences, IEEE Xplore, Hawaii, pp. 63-70, Jan 2007

[10] Channa et al., "Constraint Satisfaction in Dynamic Web Service Composition", Asian Journal of Information Technology, vol. 4, pp. 957-961, September 2005.

[11] X. Nguyen, R. Kowalczyk, and M. Phan, "Modeling and Solving QoS Composition Problem Using Fuzzy DiscCSP", IEEE Int. Conf. on Web Services (ICWS'06), IEEE CS, Chigaco, USA, pp. 55-62, December 2006.

[12] A. Hassine, S. Matsubara, and T. Ishida, "A Constraint-Based Approach to Horizontal Web Service Composition", The Semantic Web – ISWC, Springer Berlin, pp. 130-143, November 2006

[13] S. Rosario et al., "Probablistic QoS and Soft Contracts for Transaction-Based Web Services Orchestrations", IEEE Transactions on Services Computing, IEEE CS, vol. 1, pp. 187-200, October-December 2009.