Pairwise Interactions to Effectively Sample QoS in Dynamic Web Services.

Ajay Kattepur*
IRISA/INRIA
Campus Universitaire de Beaulieu
35042, Rennes-Cedex, France
Ajay.Kattepur@irisa.fr

Sagar Sen
IRISA/INRIA
Campus Universitaire de Beaulieu
35042, Rennes-Cedex, France
Sagar.Sen@irisa.fr

Benoit Baudry
IRISA/INRIA
Campus Universitaire de Beaulieu
35042, Rennes-Cedex, France
Benoit.Baudry@irisa.fr

Albert Benveniste
IRISA/INRIA
Campus Universitaire de Beaulieu
35042, Rennes-Cedex, France
Albert.Benveniste@irisa.fr

Claude Jard
ENS Cachan, IRISA
Université Européenne de Bretagne
Bruz, France
Claude.Jard@irisa.fr,

ABSTRACT

Composite web services orchestrations have the dynamic ability to choose the atomic services invoked, leading to variations in Quality of Service (QoS) behavior. To model such orchestration variability, we use the *feature diagram* (FD). Instances of the FD leads to numerous *configurations* that correspond to orchestrations of the dynamic composite service. Exhaustive exploration is impossible due to the combinatorial explosion of the configuration space, requiring efficient *sampling* of configurations. This is in order to generate a compact set of QoS values to quantify the composite services' behavior. Efficient examination of the exhaustive family of configurations by a sample satisfying *pairwise interactions* between services is selected for this purpose. Thus selected configurations are analyzed with a variety of QoS metrics to generate tradeoffs. Using the motivating examples of *car crash crisis management* and *eHealth management* case studies, it is shown that such analysis generates the entire range of QoS variations. This is in conjunction with a sampling technique that results in elimination of over 99% of configuration redundancies, while still calling all atomic services at least once. The criteria used to evaluate pairwise sampling include: a) ability to sample the extreme QoS metrics of the service; b) stable behavior of the extracted samples; c) compact set of values that can quantify QoS tradeoffs; d) comparison with random sampling. The results provide metrics to generate families of services with differing performance bounds and improved contractual description of QoS metrics.

*This work was partially funded by the ANR national research program DocFlow (ANR-06-MDCA-005), by the Région Bretagne under project CREATE ActivDoc, by INRIA under Equipe associée FOSSA and from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

Categories and Subject Descriptors

H.3.5 [**Online Information Services**]: Web-based services

1. INTRODUCTION

A *composite* web service is an application whose implementation calls other self-contained *atomic* services. A composite service *orchestration* specifies the interaction, management and coordination between the involved atomic services. Such a composite service can have dynamic runtime Quality of Service (QoS) behavior due to a number of factors: a) invocation / non-invocation of specific atomic services due to unpredictable runtime behavior; b) variable atomic services QoS; c) families of composed services that perform and behave with considerable difference in end-to-end QoS.

Such composite service behavior deteriorates contractual agreements on QoS [22]. The use of service level agreements (SLAs) [16] is the industry standard to ensure QoS compliance for both the service providers and the customers. Habitual contractual deviations of SLAs are a result of non-incorporation of QoS outliers and variable behavior of composite services. In order to increase the robustness of contractual SLAs, unpredictable behavior of composite services must be incorporated.

As such changes in QoS metrics are a result of a number of factors, we first examine the changes due to inclusion / exclusion of specific atomic services. Feature Diagrams (FD) [10] provide a formal framework to specify authorized variations in the configuration of the composite service. A dynamic composite service generally has choice to configure the invocation or rejection of available atomic services. These choices have a direct consequence on the probabilistic nature of QoS contracts [19] for the composite service. This probabilistic view of QoS captures the variable behavior of an included service's QoS.

The SLA contract has to be set for all possible configurations to effectively model the behavior of deployed composite services. However, the number of configurations grows exponentially with the number of atomic services. Consequently, it is not possible to develop a scalable approach for the definition of the composite service contract based on the exhaustive computation of QoS behavior for every

configuration. Thus, it is necessary to systematically sample the orchestration configuration space in order to analyze the QoS of web services orchestration.

In this paper, we investigate the systematic sampling of composite services configurations for the definition of a global SLA. Random sampling of configurations, generally employed, is both ineffective and expensive because it cannot be systematic and requires computing QoS values for a large number of configurations. This work focuses on the adaptation of combinatorial interaction testing [5] to select a sample of configurations that covers at least once each pairwise interaction of services. The pairwise interaction testing framework [5] provides an efficient technique to evaluate a sample space with minimal analysis. It is based on the observation that most of the faults are triggered by interactions between a small number of variables [13]. For example, consider the *car crash crisis management system* case study [12] that we will examine in this paper. With 25 optional features that may / may not be invoked in a specific orchestration, the total number of exhaustive tests required will be $33,554,432$. This is an extremely large number of tests that would require a extended time, effort and data analysis. The number of tests satisfying pairwise interaction is just 185 reducing the number of required tests by **99.99**%.

Pairwise testing has been used to detect faults in software systems in extensive prior research [5]. However, the application of these combinatorial techniques to sample configurations in a service orchestration is yet to be examined. This work performs such an examination through a series of experiments that aim at investigating the use of pairwise interaction sampling of orchestration configurations to effectively define the global SLA. This is based on a hypothesis that composite web services' QoS behavior mimics the cause of faults in software systems. Changes in QoS behavior are seen akin to the changes in interaction between "pairs" of services invoked.

The experiments are based on two case studies which are the car crash crisis management system (C^3MS) [12] and a eHealth administration system. This paper reports on the following questions:

- Q_1: For large orchestrations, it is possible to efficiently sample the behavior to generate accurate limits on QoS variability?
- Q_2: Do configurations that cover all pairwise service interactions provide such efficient sampling stratagems?
- Q_3: Are these qualitative descriptions of QoS behavior useful to generate families of orchestrations with differing SLAs?
- Q_4: How stable and effective is the global SLA computed from a pairwise configurations?
- Q_5: Is pairwise sampling more effective and robust when compared to random sampling of the configuration space?

From our experimentation, it is shown that analysis of a family of configurations (and their corresponding QoS values) can be accurately represented by a sampling configurations satisfying all pairwise interactions. The scalable approach of generating pairwise configurations can handle large orchestrations like C^3MS and eHealth scenarios while still satisfying all valid FD constraints. On comparison with the usually employed random sampling technique [7], pairwise analysis is shown to provide consistently robust results. It is also able to guarantee sufficient coverage (calling all atomic services at least once) while providing the extreme ranges of QoS variance. This leads to a compact set of QoS data that gives the composite service provider accurate / robust information to generate families of composite services. For example, a *gold* family of services should provide superior output data quality compared to say a lower costing *bronze* family of services. This comprehensive analysis of variability also helps the provider understand the global QoS extremities of the service families before negotiating a SLA agreement. Up to **80** seconds (C^3MS) and **30** seconds (eHealth) of median response time ranges are demonstrated for these services.

In a recent submission [11] [1], similar methodology is used to compare pairwise and exhaustive analysis of configuration spaces in smaller orchestrations. In this paper, that notion is extended to comparison with random runs of larger configuration spaces (where exhaustive analysis is impossible). This entails a scalable approach for robust pairwise interaction generation that is not required for the smaller example in [11]. The case studies (Section 5) and corresponding experiments (Section 6) are much larger in this paper and study the effect of not only orchestration variability, but also choice in compatible atomic service counterparts. Correspondingly, this requires further experiments on the sampling robustness and comparison with random generation (Section 6), which is not included in [11].

This paper is organized as follows. Section 2 provide foundations required for our paper. These include feature diagrams in 2.1, Orc language for orchestration in 2.2, pairwise configuration generation in 2.3, sampling configurations in 2.4 and formal description of QoS metrics in 2.5. Experimental settings are put forth in Section 3 with emphasis placed on the response time simulations in 3.1 and pairwise generation of configurations in 3.2. The methodology followed in this paper is briefly discussed in Section 4. The motivating examples of car crash crisis management system in and the eHealth administration system are explained in detail along with FD and orchestrations in Section 5. Results for the QoS analysis of the case studies are presented in 6. Comparison with respect to random generation and the stability of pairwise analysis are shown in 6.3 and 6.4, respectively. Further deliberation and perspectives of our analysis scheme are presented in Section 7. Threats to the validity of the empirical studies are discussed in Section 8. Related work in literature is put forth in section 9 followed by conclusions in section 10.

2. FOUNDATIONS

Described here are some fundamental descriptions required to understand the rest of the paper.

2.1 Feature Diagrams

Feature Diagrams (FD) introduced by Kang et al. [10] compactly represent all the products (referred to as *configurations* in this paper) of a software product line (SPL) in terms of features which can be composed. Feature diagrams have been formalized to perform SPL analysis [21]. In [21], Schobbens et al. propose a generic formal definition of FD which subsumes many existing FD dialects. We define a FD as follows:

- A FD consists of k features $f_1, f_2, ..., f_k$
- Each feature f_i may be associated with a software asset such as an atomic service.
- Features are organized in a parent-child relationship in a tree T. A feature with no children is called a leaf.
- A parent-child relationship between features f_p and f_c are categorized as follows:

[1] http://www.irisa.fr/triskell/members/sagarsen/papers/submitted/ICWS2010

- *Mandatory* - child feature f_c is required if f_p is selected.
- *Optional* - child feature f_c *may* be selected if f_p is selected.
- *OR* - at least one of the child-features f_{c1},f_{c2},..,f_{c3} of f_p must be selected.
- *XOR* - one of the child-features f_{c1},f_{c2},..,f_{ck} of f_p must be selected.

- Cross tree relationships between two features f_i and f_j in the tree T are categorized as follows:
  - f_i requires f_j - The selection of f_i in a product implies the selection of f_j.
  - f_i excludes f_j - f_i and f_j cannot be part of the same product and are *mutually exclusive*.

Using the FD we create and validate configurations (i.e a selection of features in the FD) of atomic services.

2.2 Service Orchestrations using Orc

While the FD describes variable paths available in services' interactions, it is crucial to formally describe the composite orchestrations to avoid ambiguities during deployment. Orc [15] serves as a simple yet powerful concurrent programming language to describe web services orchestration behavior.

The fundamental declaration used in the Orc language is a *site*. Sites can be both *external* or *internal*. The type of a *site* is itself treated like a service - it is passed the types of its arguments, and responds with a return type for those arguments. An Orc *expression* represents an execution and may call external services to publish some number of values (possibly zero).

Orc has the following combinators that are used on various examples as seen in [15]. The *Parallel* combinator $F|G$, where F and G are Orc expressions, runs by executing F and G concurrently. Whenever F or G communicates with a service or publishes a value, $F|G$ does so as well. The execution of the *Sequential* combinator $F > x > G$ starts by executing F. Sequential operators may also be written compactly as $F \gg G$. Values published by copies of G are published by the whole expression, but the values published by F are not published by the whole expression; they are consumed by the variable binding. If there is no response from either of the sites, the expression does not terminate. While the above two composition operators are for creating threads, Orc uses the following construct to prune operations. The *Pruning* combinator, written $F < x < G$, allows us to block a computation waiting for a result, or terminate a computation. The execution of $F < x < G$ starts by executing F and G in parallel. Whenever F publishes a value, that value is published by the entire execution. When G publishes its first value, that value is bound to x in F, and then the execution of G is immediately terminated. The *Otherwise* combinator, written $F;G$ has the following execution. First, F is executed. If F completes, and has not published any values, then G executes. If F did publish one or more values, then G is ignored. The publications of $F;G$ are those of F if F publishes, or those of G otherwise. In the *Fork-Join* combinator, two processes are invoked and run concurrently. The process waits until a response is obtained from both. This may be represented as (F, G) where the process waits for responses from both atomic services F and G.

The FD and the orchestration cover two dimensions that are complementary to each other. While the FD represents the variability in the configurations, the orchestration specifies the order in which the services are called. Making use of the terminology in [21], *primitive* features are "features" that are of interest and that will be incorporated in real-world services. On the contrary, *decomposable* features are just intermediate nodes used for decomposition. It is up to the modeler to determine such classification of features in the FD. We extend the semantics given in [21] to ensure compatibility of an orchestration with the feature model as follows:

- The set of available services S are the *primitive* nodes of the FD D;
- For each orchestration, the set of corresponding services invoked (denoted N);
- $N \subseteq S$ in a configuration;
- A model of D is a subset of its (*primitive* and *decomposable*) nodes;
- There must exist a model of D ($[[D]]$) such that $[[D]] \cap S = N$ (a model of a FD is a subtree that is valid w.r.t. the operators and the dependence relation).

Drawing from the real-world services and the constraints shown in a FD, the composite service may be developed by an orchestrator.

2.3 Configuration Generation

Combinatorial interaction testing (CIT) has been proposed by Cohen et al. [5] to select a subset of all combinations of variables that define the input domain of a program, while still guaranteeing a certain level of coverage. This has led to the definition of pairwise interaction testing, or 2-way testing. This samples the set of all combinations in such a way that all possible pairs of variable values are included in the set of test data. Pairwise testing has been generalized to t-way testing which samples the input domain to cover all t-way combinations.

DEFINITION. 1. ***Covering Array*** - *A covering array CA $(N; t, k, v)$ is a $N \times k$ array on v symbols with the property that every $N \times t$ sub-array contains all ordered subsets of size t from v symbols at least once.*

From the definition of a covering array, the strength t of the array is the parameter that allows achieving 2-way (pairwise), 3-way or t-way combinations. The k columns on this array correspond to all the variables in the input domain. For the generation of services configurations, k is the number of services, and v is 2 since we have only boolean variables (services may be present or absent in a configuration). The problem of generating a minimal covering array for a set of variables is a complex optimization problem that has been studied in extensive prior work for example [5]. It is important to notice that there exist very few studies that have tackled the automatic generation of CIT in the presence of constraints between variables [6]. In order to include properties that forbid combinations of values, CIT generation techniques have to allow the introduction of constraints in the algorithms that generate covering arrays. We have developed a solution to generate t-wise configurations that satisfy all constraints modeled in a feature model [18]. This solution is based on the Alloy analyzer and SAT solving.

As the CIT removes redundant samples, there are a myriad of sets of configurations that satisfy all the pairwise constraints. So, there are many sets of pairwise configuration solutions (referred to a *samples* from now on) that exist for a particular feature diagram. The consistency of these samples of solutions must be tested to determine the accuracy and stability in selecting pairwise combinations.

2.4 Sampling Configurations

As stated in [13], if all faults in a system can be triggered by a combination of n or fewer parameters, then testing all n-tuples of parameters is effectively equivalent to exhaustive testing. This hypothesis is extended to web services' QoS, effectively sampling n-tuples of services to represent the exhaustive range of service interactions.

In order to demonstrate this, consider the set of four atomic services (A, B, C, D) with varying response times. The atomic services can be composed in 2^4 exhaustive combinations. However, if we consider the service combinations in pairs, we require fewer configurations. These can be subsumed by 6 sets of configurations that cover these pairs of interactions resulting in removal of 62.5% of redundancies. This is shown in Table 1 where, for example, interaction (A, B) refers to calling both service A and B while (A, ¬B) refers to calling only A with B explicitly not invoked. Such

Pairwise Interaction	Configurations
(A, B); (A, C); (A, D); (B, C); (C,D)	(A, B, C, D)
(A, ¬B); (A, ¬C); (A, ¬D)	(A)
(B, D); (B, ¬A); (B, ¬C); (D, ¬A)	(B, D)
(C, ¬A); (C, ¬B); (C, ¬D)	(C)
(D, ¬B); (D, ¬C)	(A, D)
(B, ¬D)	(A, B, C)

Table 1: Subsuming pairwise interactions in configurations.

an intuitive procedure of sampling configurations is invaluable for larger composite services, that may be composed in many thousands of ways. Pairwise sampling these composite services provides an accurate portrayal of overall QoS deviation, while eliminating redundant configurations (that have no significant contribution to the global QoS variance).

2.5 QoS Aspects of the Orchestration

The use of hard contracts to regulate QoS parameters such as response time, availability and so on has been the norm for most SLAs. However these take into account many outliers that are the result of some rare deviations in QoS which generate pessimistic SLAs. Probabilistic analysis of QoS parameters as shown in [19] [8] provides a more realistic study of actual web services' behavior. The following QoS parameters have been chosen for our study:

1. *Latency / Response Time* (T) - Denotes the overall delay due to the time taken by a web service to respond. It is a discrete value that may be modeled as a long tailed distribution incorporating some *rare deviations*.

2. *Availability* (α) - The probability that a service is active and can respond to a service call. For a well managed service, this value is generally quite high.

3. *Cost* (χ) - Refers to the monetary cost associated with each invocation of a particular atomic service.

4. *Data Quality* (ξ) - A subjective measure of trade off to high Cost and Response times of web services. It measures the "Quality" of the output of the web service and the beneficial aspects of including a new atomic service into the composite orchestration.

Extending these QoS parameters to an orchestration involves the use of Orc combinators as described previously. Taking two sites s_i and s_j, the QoS parameters may be applied as shown in Table 2 depending on the Orc combinators used. The cases of composing the service s_{ij} using the *sequential* and *fork-join* combinators have been considered. The latency, cost and availability metrics for the composite service s_{ij} are derived as shown in [4] with $Max(p, q)$ representing the maxima of the values p and q. For the sequential case, the latency and cost of the composite service is a sum of the atomic services' parameters while the availability is a product of such parameters. Similarly, the maxima of the atomic services' response times contributes to the global response time under parallel invocation.

Orc Expression	$s_{ij} \triangleq s_i \gg s_j$	$s_{ij} \triangleq (s_i, s_j)$
Latency	$T(s_{ij}) = T(s_i) + T(s_j)$	$T(s_{ij}) = Max(T(s_i), T(s_j))$
Cost	$\chi(s_{ij}) = \chi(s_i) + \chi(s_j)$	$\chi(s_{ij}) = \chi(s_i) + \chi(s_j)$
Availability	$\alpha(s_{ij}) = \alpha(s_i) \times \alpha(s_j)$	$\alpha(s_{ij}) = \alpha(s_i) \times \alpha(s_j)$

Table 2: QoS metrics extended to Orc combinators.

3. EXPERIMENTAL SETUP

The experiments involved simulating probabilistic QoS behavior of atomic services and pairwise generation of configurations.

3.1 Simulation of QoS Distributions

The first step is simulating the probabilistic response time distributions of each atomic web service as done in [19]. For this, we make use of the *t-location distribution* fitting feature in MATLAB as shown in Fig. 1. By varying the degrees of freedom ν and non-centrality parameter δ in the *dfittool* of MATLAB, it is possible to generate various heavy tailed distributions that mimic the response times of web services. These are used to simulate the response times of actually invoked atomic services. This t-distribution fitting was used

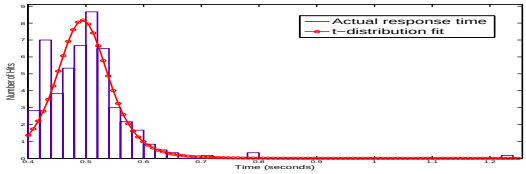

Figure 1: Distribution fitting of actual response times of a web service invocation.

to generate various distributions of services' response times with varying parameters.

3.2 Generating a sample of pairwise configurations

We transform the feature diagram to constraint satisfaction problem model in the language Alloy as described in [18]. The features in the FD are transformed to concepts in Alloy called *signatures*. Inter-feature constraints in the FD are transformed to Alloy *facts*. All pair-wise interactions between features are transformed to Alloy *predicates*. The goal of solving the Alloy model is to find the minimal set of configurations that cover conjunctions of all valid pair-wise predicates. The first step involves *detection* of all valid pairs that conform to the FD. In the second step, we construct conjunctions of pair-wise predicates and solve them via incrementally increasing the scope of the solution size. The result is a minimal set of configurations that cover conjunctions of all valid pairs. At times the SAT solver in Alloy is not scalable for a large FD. We apply divide-and-compose approaches as described in [18] to handle this scalability issue.

While this view makes use of static invocation of an orchestration (based on the FD configurations), another view is also possible. Dynamic invocation of the FD (with degraded modes) by a changing orchestration. In this case, the dynamic orchestration output determines which configuration of the FD is invoked. However, in the current implementation, we resort to static invocation of services dependent on variables.

4. METHODOLOGY FOR EXPERIMENTATION

We present a methodology designed to examine the questions that we had developed in Section 1. The following steps summarize our methodology, with corresponding questions in brackets:

1. The modeling inputs may be specified as a 3-tuple (S, FD, O) where:
   (a) S is the set of services that can be used. In a configuration, subsets $S_1, ..., S_N$ of these services are used.
   (b) FD is the constraints for the services included in a particular configuration.
   (c) O is the set of orchestrations $O_1, ..., O_M$ in a composite service. These orchestrations invoke the services $S_1, ..., S_N$ according to the configuration constraints specified by the FD.
2. The CIT with pairwise constraints satisfied is then used to sample a set of configurations from the FD. This represents a subset of configurations that effectively cover all the exhaustive configurations in the FD (Q_1, Q_2).
3. For each of the sampled configurations we compute the QoS for orchestrations ($\subset O$) invoking all atomic services in the configuration, at least once. These include a set of parameters to analyze tradeoff between atomic services' inclusion / deletion between configurations. Probabilistic models of response time are used to provide an accurate portrayal of the services' behavior to be traded-off with other QoS metrics (Q_3).
4. Comparisons with randomly generated configurations and consistency over multiple sample sets are included to experimentally study the robustness of the proposed pairwise analysis scheme and corresponding QoS metrics (Q_4, Q_5).

5. CASE STUDIES

Described in this section are the feature diagrams and orchestrations of two case studies.

5.1 Car Crash Crisis Management System Case Study

The need for crisis management systems has grown significantly over time [12]. A crisis can range from major to catastrophic affecting many segments of society. Crisis management involves identifying, assessing, and handling the crisis situation. A crisis management system facilitates this process by orchestrating the communication between all (distributed) parties involved in handling the crisis. The car crash crisis management system (C^3MS) [12] includes all the functionalities of a general crisis management systems, and some additional features specific to car crashes such as facilitating the rescuing of victims at the crisis scene and the use of tow trucks to remove damaged vehicles. As described in [12], the main goals of this system include: a) Facilitating the rescue mission carried out by the police / firemen and providing them with detailed information on the location of the crash. b) Managing the dispatch of ambulances or other alternate emergency vehicles to transport victims from the crisis scene to hospitals. c) Coordinating the first-aid missions by providing relevant medical history of identified victims by querying data bases of local hospitals. d) Ushering the medical treatment process of victims by providing important information about the crash to the concerned workers. e) Managing the use of tow trucks to remove obstacles and damaged vehicles from the crisis scene.

5.1.1 Feature Diagram of Car Crash Crisis Management System

In Figure 2, we present the Car Crash Crisis Management System (C^3MS) FD [12]. The C^3MS FD contains several features that are associated with software assets represented by atomic services. For example, the *Paramedic* feature is represented by the *Paramedic* web service. Some sets of features like *Police* and *PoliceMan* are subsumed by a single service *Police*. Constraints such as optional, requires and mutual exclusion (XOR) are also incorporated. For example, the *GPS* and *GSM* features are mutually exclusive while the *Doctor* feature requires the *PublicHospital* feature.

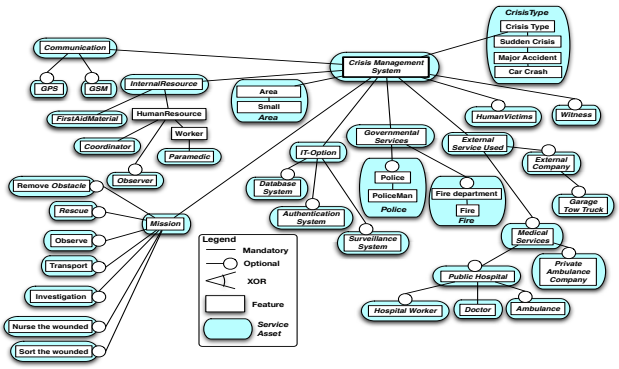

Figure 2: C^3MS Feature Diagram.

5.1.2 Service Orchestrations in Car Crash Crisis Management System

A host of web services may be used to represent the C^3MS. The corresponding web services referring to the features are shown in 3. It is assumed that the web services perform the functions as generally specified by the nomenclature. For example, the *CommunicationManager* service manages the communication between parties while the *Ambulance* service regulates ambulances to the car crash sites. These abstract representation of web services do not inherently require working details and can be modified to generic specifications. Their construction may be modified according to specifications to perform/subsume other associated tasks.

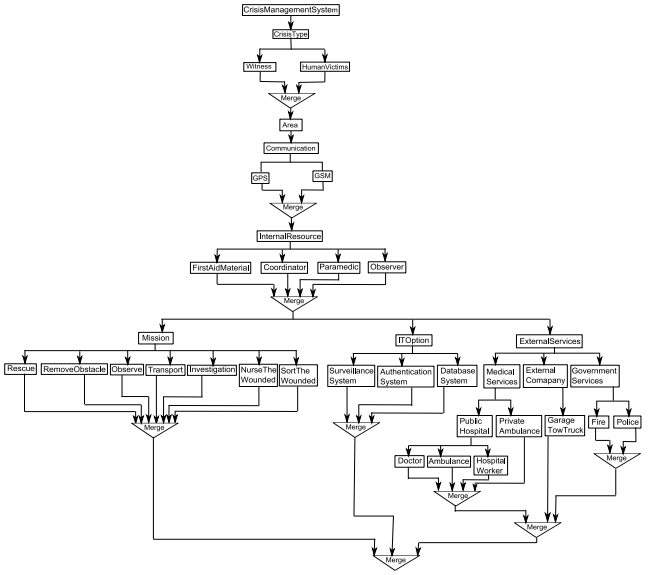

Figure 3: Composite Web Service Orchestration of the C^3MS.

$CrisisManagementSystem() \triangleq CrisisType() \gg (HumanVictims(h),\ Witness(w)) \gg Area() \gg CommunicationManager(gs,gp) \gg InternalResource(o) \gg (Mission(),\ ITOption(IT),\ ExternalServices(),\ Medical(md))$

$CommunicationManager(gs,gp) \triangleq (GSM(gs),GPS(gp))$
$InternalResource(o) \triangleq (AidMaterial(),Coordinator(),Paramedic(),\ Observer(o))$
$Mission() \triangleq (RemoveObstacle(robs),Rescue(re),Observe(ob),\ Transport(tr),Investigation(in),NurseWounded(nw),SortWounded(sw))$
$ITOption(IT) \triangleq (su,au,db) \gg (SurveillanceSystem(su),AuthenticationSystem(au),\ DatabaseSystem(db))$
$ExternalServices() \triangleq (ExternalCompany(ec),GovernmentServices())$
$Medical(md) \triangleq (pac,ph) \gg (PrivateAmbulance(pac),PublicHospital(ph))$

$ExternalCompany(ec) \triangleq \mathtt{tt} \gg TowTruck(tt)$
$GovernmentServices() \triangleq (Police(pm),Fire(fm))$
$PublicHospital(ph) \triangleq (hw,amb,doc) \gg (HospitalWorker(hw),Ambulance(amb),Doctor(doc))$

Table 3: Orc pseudo code of the C^3MS orchestration.

The composite service orchestration is represented succinctly in Fig. 3 and the Orc pseudo code is presented in Table 3. They represent a family of configurations that may be invoked. Calling the *CrisisManagementSystem()* service invokes other services like *CrisisType* and *InternalResource()* operations in sequence. These services, in turn, may call other services in parallel or services, passing some parameters in the process. For instance, the *CommunicationManager()* services calls the *GPS()* and *GSM()* services in parallel, while passing some parameter values for invocation of these services. The setting of these parameter results in the various associated configurations in the system. Operations such as mutual exclusion (MUX) and synchronization ($Merge$) may be performed using Orc constructs. Another level of control is the global timeout value associated with the composite service. This has to be associated with the overall SLA of the composite service to provide optimal durations for response.

5.2 eHealth Management System

The need for efficient hospital management stems has been discussed in [20]. A hospital administration system is devised to remove some of the inefficiency plaguing current protocols such as cumbersome admission time, duplicate data entry, redundant lab tests, ineffective treatment coordination, and billing processes. Drawing inspiration from [20] composite health care applications are required to connect various parties and locations. The information flows seamlessly across organizational and system boundaries emitting from the use of such a centralized orchestration. This enhanced visibility gives everyone involved a unified view of relevant information and gives process owners the ability to improve existing methods and procedures. The eHealth system can be viewed as an extension of the C^3MS medical services to transport injured victims for speedy treatment of injuries. Examples of the utility of healthcare applications include: a) Healthcare providers can access the medical information of a prospective patient and use ambulance services to transfer the client to relevant healthcare facilities. b) Physicians can review a patient's medical history even though this data resides in several systems managed by diverse providers. c) Insurance claims and financial options can be updated and handled in a speedy way. d) Doctors can use a composite application to determine the appropriate medication for a patient, order the drug, check the status of pharmacy approval, and monitor how the drug is dispensed. e) Special needs of the patient such as catering specific food items and lab tests can be coordinated in an effective way.

5.2.1 Feature Diagram of eHealth System

Fig. 4, presents the eHealth management system FD. Similar to the C^3MS FD, it contains several features that are associated with software assets represented by atomic services. Constraints such as optional, requires and mutual exclusion (XOR) are also incorporated. Two versions of the similar service $Ambulance_f$ and $Ambulance_s$ are in mutual exclusion. These atomic features or services can be set to varying QoS values resulting in interesting combinations of services.

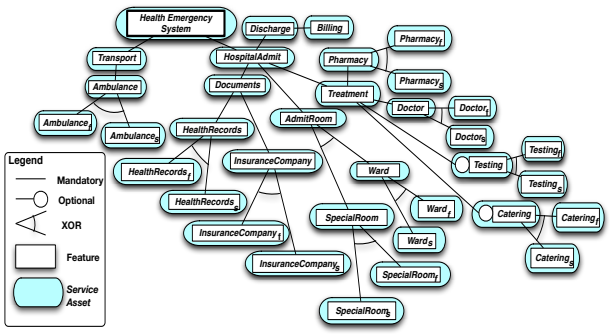

Figure 4: eHealth Feature Diagram.

5.2.2 Service Orchestrations of eHealth System

The web services used for the orchestration of the eHealth system are shown in Table 4. The operations are generic with services such as *HealthRecords* and *InsuranceRecords* used to request relevant medical history and insurance status of the patient, respectively. The Orc pseudo code for the

$HealthEmergencySystem() \triangleq Transport() \gg HospitalAdmit() \gg Billing() \gg Discharge()$

$Transport() \triangleq a \gg Ambulance(a)$
$HospitalAdmit() \triangleq Documents() > (hf,in) > (HealthRecords(hf),\ InsuranceRecords(in)) \gg AdmitRoom() \gg Treatment()$
$AdmitRoom() \triangleq (sr,w) \gg (SpecialRoom(sr),Ward(w))$
$Treatment() \triangleq (d,t,c,p) \gg ((Doctor(d),Testing(t),\ Catering(c),Pharmacy(p))$

$Ambulance(a) \triangleq \mathtt{let}\ (Ambulance_f() \mid Ambulance_s())$
$HealthRecords(hf) \triangleq \mathtt{let}\ (HealthRecords_f() \mid HealthRecords_s())$
$InsuranceRecords(in) \triangleq \mathtt{let}\ (InsuranceRecords_f() \mid InsuranceRecords_s())$
$SpecialRoom(sr) \triangleq \mathtt{let}\ (SpecialRoom_f() \mid SpecialRoom_s())$
$Ward(w) \triangleq \mathtt{let}\ (Ward_f() \mid Ward_s())$
$Doctor(d) \triangleq \mathtt{let}\ (Doctor_f() \mid Doctor_s())$
$Testing(t) \triangleq \mathtt{let}\ (Testing_f() \mid Testing_s())$
$Catering(c) \triangleq \mathtt{let}\ (Catering_f() \mid Catering_s())$
$Pharmacy(p) \triangleq \mathtt{let}\ (Pharmacy_f() \mid Pharmacy_s())$

Table 4: Orc pseudo code of the eHealth orchestration.

eHealth system is presented in Table 4. The distinguishing feature of this orchestration is the choice of services that can be used to perform similar goals. For instance, either one of the mutually exclusive (MUX) services $Testing_f()$ or $Testing_s()$ services can be used to request for lab tests. However, the QoS associated with each of these services is different resulting in varying overall composite service QoS.

6. EXPERIMENTS ON QOS EVALUATION

The efficacy of the offline analysis procedure was tested experimentally for both the case studies.

6.1 Evaluating QoS of the Car Crash Crisis Management System

The response time distributions for the atomic services were randomly assigned. The settings for the range of t-distributions in MATLAB included degrees of freedom ν for 3 to 8 and non-centrality parameter δ from 5 to 15 seconds, respectively. These values were chosen to provide diversity in atomic response time values and can be varied according to the service implementation. For an included service, the individual timeout value was set to 95 percentile of the response time distribution. The global timeout value was also set sufficiently high (300 seconds) to allow capture of outliers in the distribution. For each chosen configuration, $\mathbf{10,000}$ Monte-Carlo runs on the chosen services in the orchestration (representing a partial order of the composite service) was run. The response time of the orchestration was collected during each run to generate an associated distribution.

The approach presented in [18] was used to generate a minimal set (given the resource constraints) of configurations that satisfy pairwise interaction. The input settings to the configuration generator are (a) Maximum scope for Alloy solver (b) Maximum time to solve (c) Divide-and-compose strategy for scalable generation. The maximum scope is set to 8 and maximum time to 2000 milli-seconds with use of *incremental growth* strategy. Through this technique, 185 configurations for the C^3MS case study were generated. The 185 configurations satisfy all valid pairwise interactions between services in the C^3MS FD that originally specify 2^{25} configurations. All invalid pairs that do not conform to the FD are rejected by the approach. For instance, the not including the *Mission* feature in a configuration is invalid as it is a mandatory feature.

As seen in Fig. 5, the pairwise generated configurations cover a range of response time distributions. The distributions were sorted in increasing order of response time and are shown. The slowest and the fastest composite services are marked. Their median values are shown to be **113** and **201** seconds, respectively. This demonstrates the use of a few configurations to test significant changes of about **88** seconds response time in a composite service. This answers the questions Q_1 and Q_2 in Section 1, that pertain to the efficacy of pairwise sampling to generate a range of output QoS values.

In Fig. 5, the extreme configurations have been marked. This is further seen in the *box-plot* representation in Fig. 6. On each box, the central mark is the median, the horizontal edges of the box are the 25th and 75th percentiles, the whiskers extend to the most extreme data points (not considered outliers) and outliers plotted individually. The boxplot captures the minima, 25, 50, 75 and 95 percentile values of a configuration's response time distribution. 37 of the distributions (arranged from fastest to slowest) are provided in the box-plot representation. The median values of the extreme configurations are once again marked by horizontal lines.

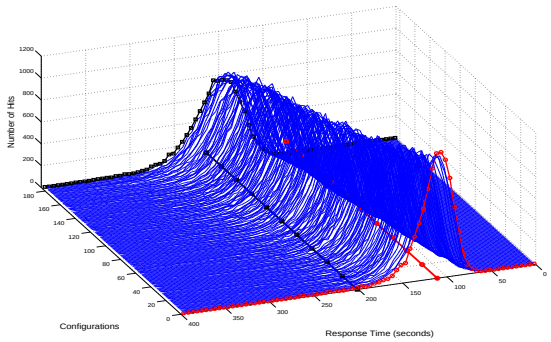

Figure 5: Response time distributions of the 185 pairwise configurations for C^3MS.

Additional parameters such as availability of a service, the cost entailed in calling atomic services and output data quality is also be studied in tandem. Using the combinators described in Table 2, the QoS parameters were analyzed for each configuration generated by the pairwise interactions. Setting atomic service availability to **0.99** (representing service availability in 99% of invocations) the composite availability of each configuration is shown in Fig. 7. The output data quality ξ is related to the cost χ by the constant κ given by $\xi = \chi/\kappa$ (assuming linear increase in data quality with each atomic service invocation). The output data quality

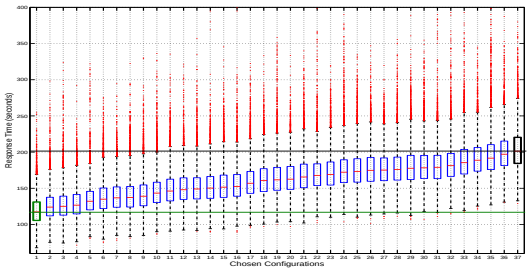

Figure 6: Box-plot representation of the 37 selected pairwise configurations for C^3MS.

ξ is can also be derived exponentailly from the cost χ by $\xi = e^{\chi/\kappa}$. For example, setting the $\chi = 5$ units for each invoked atomic service, the cost of each configuration is shown in Fig. 7. Furthermore, setting $\kappa = 20$, the linear and exponentail output data quality of the configurations may also be derived. Though simplistic in outlook (due to subjectivity of cost and data quality), this trade-off of parameters must be taken into account. These myriad of QoS parameters accurately quantify run-time behavior of the composite service. This is necessary to evaluate to answer the question Q_3 in Section 1 referring to generation of families of composite services.

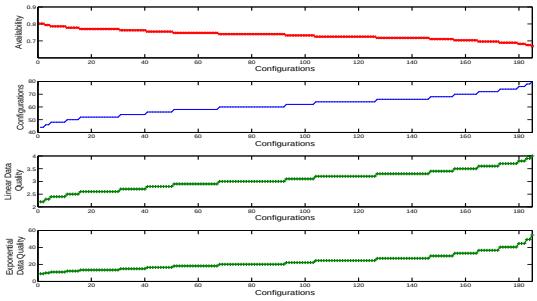

Figure 7: Availability, Data Quality and Cost of the pairwise configurations of C^3MS.

6.2 Evaluating QoS of the eHealth System

The settings for the range of t-distributions in MATLAB included degrees of freedom ν for 3 to 8 and non-centrality parameter δ from 5 to 15 seconds, respectively. For the faster web services (marked with the subscript $_f$), the δ parameter was set between 3 to 5 seconds, representing a faster response to a service call.

Similar to the C^3MS case, 188 configurations were deemed sufficient by the solver to satisfy all valid pairwise interactions from a total set of 2^{12} configurations. For each chosen configuration, $\mathbf{10,000}$ Monte-Carlo runs on the chosen services in the orchestration was performed. The response time of the orchestration was collected during each run to generate an associated distribution.

As seen in Fig. 8, the pairwise generated configurations cover a range of response time distributions. The distributions were sorted in increasing order of response time and are shown. The slowest and the fastest composite services are marked with median values. In the case of eHealth, the 30 seconds range in response time values is due to the added diversity of *choice* in choosing a *fast* or *slow* atomic service. This further answers the questions Q_1 and Q_2 in Section 1. Using the combinators described in Table 2, the QoS parameters were analyzed for each configuration generated by

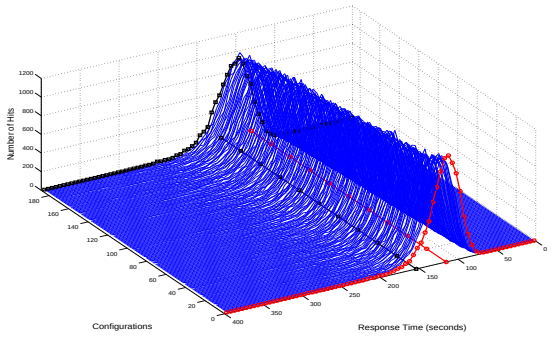

Figure 8: Response time distributions of the 188 pairwise configurations for eHealth.

the pairwise interactions. Setting atomic service availability to 0.99 the composite availability each configuration is shown in Fig. 9. The cost of the service is varied depending on the faster or slower service used. A faster service (with subscript $_f$) is set *double* the cost of its slower (with subscript $_s$) counterpart. This changes the range of cost and data quality available for different configurations as seen in Fig. 9.

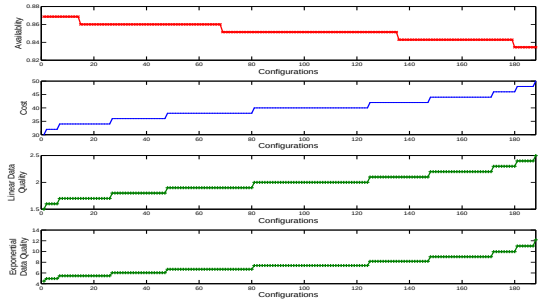

Figure 9: Availability, Data Quality and Cost of the pairwise configurations of eHealth.

6.3 Comparison with Random Sampling

It has been shown in [5] that pairwise interaction testing of such configurations is advantageous over random sampling of the sample space due to efficient coverage and greater control over systematic testing. With random runs, it is impossible to determine if all the atomic services have been invoked at least once. The configurations leading to extreme test case values need not be necessarily generated during random runs and there may be many redundant configurations invoked repeatedly. Setting SLAs based on random runs is both non-robust and can lead to habitual deviance. Generating families of configuration with accurately fixed bounds on QoS is also not possible. For these reasons, pairwise generation has comparative advantages over random runs. The pairwise setting ensures that every atomic service is invoked at least once in the sample.

Three sets of random configurations were generated as shown in Fig. 10, each with original configuration size 185. In each case, the number of valid configurations was found to be 17, 21 and 24 resulting in a maximum efficient generation percentage of 12.97%. Not only are there deviations in the number of valid configurations for each run (17, 21, 24), but also in the QoS metrics output in each run. SLA deviations are a result of resorting to such insufficient random runs of a composite service, which might generate invalid and redundant scenarios. To test the efficacy of combinatorial testing the **185** pairwise configurations were compared with random samples for the C^3MS. All the mandatory features were set to be invoked with the constrained and optional features randomized in invocation for the random case. This random sampling was performed by a Markov decision process of traversing features in the FD, which will always lead to generation of valid configurations (based on constraints). The comparison with pairwise is shown in Fig. 11 and it is seen that random generation can cover a large range of QoS values if sufficient number of configurations are generated. To determine that number, however, requires analysis of pairwise interactions. The random configurations are deficient as they cannot guarantee a) invocation of every possible service at least once; b) generating the extreme configurations for a particular composite service in every sample. When compared to the pairwise generation scheme that covered all pairs of services, the random generation covered only **8.8**% of the service pairs. This shows that the same set of services are redundantly invoked in many configurations during random generation. Thus, for such orchestrations

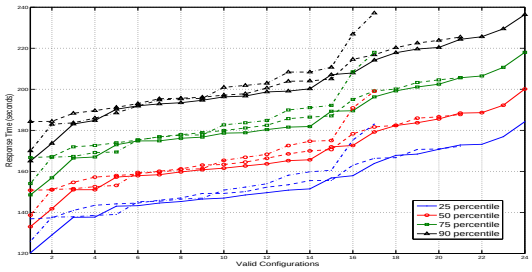

Figure 10: Three runs of random generation of configurations for C^3MS.

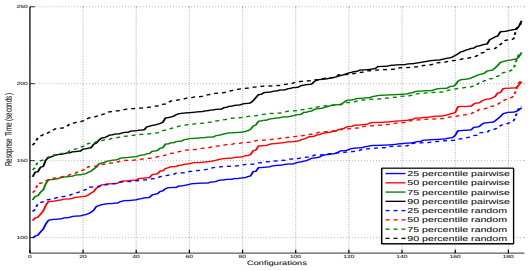

Figure 11: Comparison of pairwise and random response time (arranged in increasing order) of percentile values for 185 configurations of C^3MS.

with numerous configurations, using pairwise interactions is a sufficient choice in order to examine the entire sample space. This answers the question Q_5 in Section 1, referring to the comparison between pairwise and random sampling.

6.4 Consistency of Pairwise Samples

Given one orchestration, there can be many different sets of configurations that cover pairwise services interactions. Thus, we compute QoS behavior over different samples of configurations. This aims at evaluating the stability of pairwise interaction coverage as a sampling heuristic to estimate the global QoS for an orchestration. A collection of **10** samples that satisfy the pairwise interaction testing were generated for the eHealth case. The percentile statistics of the configurations in each sample was collected through 10,000 Monte-Carlo runs and is shown in Fig. 12. The lowest and

Percentile	25(min.)	25(max.)	50(min.)	50(max.)	75(min.)	75(max.)	90(min.)	90(max.)
Pairwise Standard Deviation(seconds)	2.18	1.52	2.59	1.73	2.90	1.82	3.19	1.83
Random Standard Deviation(seconds)	4.14	4.17	4.21	4.51	4.43	4.76	4.63	5.07

Table 5: Standard Deviation values for pairwise and random samples.

highest percentile values of the configurations in each sample were collected. The mean inter-sample difference for the random case is **12.94** seconds compared to **6.44** seconds for the pairwise case. Further, these were compared with **10** samples of randomly generated configurations (with 300 configurations in each sample) in Fig. 12. Again, all the mandatory features were set to be invoked with the constrained and optional features randomized in invocation for the random configurations. The number of valid configurations for each sample ranged between 3.5% to 9% of the 300 configurations. Comparing the two cases, the stability of the pairwise generation is demonstrated through its consistently low standard deviation values in Table 5 when compared to random samples. Once again, the lowest and the highest percentile values of all the configurations in a particular sample are compared. This answers the question Q_4 in Section 1, referring to the stability of pairwise sampling.

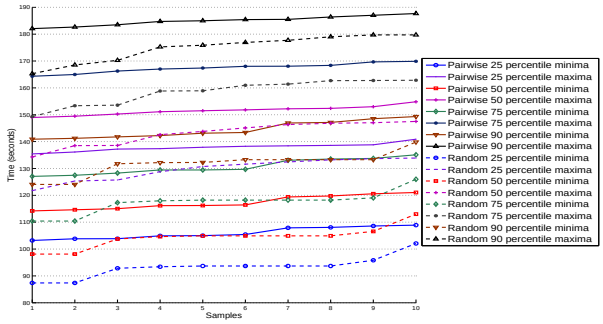

Figure 12: Comparing stability of pairwise and random samples for eHealth.

7. PERSPECTIVES DUE TO ANALYSIS

The methodology evaluated for the C^3MS and the eHealth orchestrations can lead to many possibilities for improving QoS metrics for composite services. This includes setting the SLA keeping into account the worst performing configuration. This will prevent contract deviation during actual deployment of the service.

A family of SLAs for a set of configurations taking into account trade-offs between QoS metrics and the output quality of configurations may be proposed. This leads to families of composite services with extensively analyzed SLAs. Configurations may be grouped along with their QoS behavior to develop an extended product line of composite services. For example, categories of services may be constructed for the C^3MS orchestration (based on Figs. 5 and 7) as shown in Table 6. Similarly, the two categories of service families for the eHealth case (Figs. 8 and 9) is shown in Table 7. In both cases, the family of services with higher data quality is traded-off by a slightly higher response time.

While the diversity in QoS families for the C^3MS is due to optional services that may / may not be included, the variability in the eHealth case is mainly due to other factors. An inherent choice in replacing a *slow* atomic service with a *fast* counterpart can lead to a range of QoS values. Generated configuration families can use of combination of these options of optimally compose atomic services to specific QoS bounds. These service families can have associated contracts (albeit in the soft-sense as in [19]) to monitor deviations from specifications. These instances provide answers to the question Q_3 in Section 1, that pertains to developing families of composite service orchestration with significantly different QoS behavior. With numerous possible combinations of atomic services, such a dedicated families of services with significantly different QoS outputs enable accurate monitoring of services provided. The pairwise scheme is both a robust and compact representation of the behavior space of the set of orchestrations. This provides an effective pre-SLA technique to enunciate the QoS metrics and threshold levels.

Configuration Families	Bronze	Silver	Gold
90 percentile Response Time (T)	< 183 s	< 216 s	≥ 216 s
Median Response Time (T)	< 150 s	< 179 s	≥ 179 s
Availability (α)	> 0.75	> 0.71	≥ 0.71
Cost (χ)	< 60	< 70	≥ 70
Linear Data Quality (ξ)	< 3	< 3.5	≥ 3.5
Exponential Data Quality (ξ)	< 20	< 30	≥ 30

Table 6: Configuration families for C^3MS.

Configuration Families	Standard	Premium
90 percentile Response Time (T)	< 171 s	≥ 171 s
Median Response Time (T)	< 139 s	≥ 139 s
Availability (α)	> 0.85	≤ 0.85
Cost (χ)	≤ 40	> 40
Linear Data Quality (ξ)	≤ 2	> 2
Exponential Data Quality (ξ)	≤ 8	> 8

Table 7: Configuration families for eHealth.

8. THREATS TO VALIDITY

This section considers the threats to the validity of the experimental results. These may be *internal* (whether there is a bias/error in the experimental design which could affect the causal relationship) or *external* (ability to generalize the results of the experiment to industrial practice).

The hypothesis studied in this paper concerns the use of pairwise sampling to evaluate QoS of large orchestrations. Sources of internal error can be a result of the MiniSAT solver used to generate the pairwise configurations or the MATLAB statistical tools used for QoS evaluation. These tools have not been compared with available alternatives for consistency of results. Furthermore, the assumption is that for each sample of configurations, the pairwise analysis scheme can provide consistently large range of QoS values. Systematic bias in QoS may be introduced in samples when extreme cases are not generated.

To ensure scalability to large industry level FDs, the pairwise generation in [18] makes use of incremental growth / binary splitting schemes. Redundancies in the number of configurations can be seen due to these schemes. For generating more than one sample of solutions, the symmetry breaking scheme in Alloy was used. This introduces more constraints with each proceeding sample, which increases the time required to generate such samples.

9. RELATED WORK

The combinatorial testing framework described by Cohen et al. [5] has been applied extensively to efficient testing for fault detection. In the work of Cohen et al. [6], this technique is extended to software product lines with highly configurable systems. Modeling variability in SPLs using feature models is the work of Jaring and Boschet [9] where they show that the robustness of a SPL architecture is related to the type of variability. To ensure that constraints in the FD are incorporated in the efficient sampling of t-wise tests, the scalable solver proposed by Perrouin et al. [18] is used. In [14], variability in software as a service applications are modeled using the orthogonal variability model to study the customization choices in such workflows.

Pre-deployment testing of SLAs has been studied by Di Penta et al. [17], where they make use of genetic algorithms to generate test data causing SLA violations. Analysis of white and black box approaches are provided in the paper. In [2], Bruno et al. make use of regression testing to ensure that an evolving service maintains the functional and QoS assumptions. The service consistency verification due to evolution is done by executing test suites contained in a XML encoded facet attached to the service.

The use of probabilistic QoS and soft contracts was introduced by Rosario et. al [19] and Bistarelli et al. [1]. Instead of using fixed hard bound values for parameters such as response time, the authors proposed a soft contract monitoring approach to model the QoS measurement. The composite service QoS was modeled using probabilistic processes by Hwang et al. [8] where the authors combine orchestration constructs to derive global probability distributions.

In our paper, we extend these two notions to analyze the QoS of a composite orchestration under various configurations. Effective sampling of orchestrations is necessary specially in conjunction with exceedingly flexible and large configuration spaces. When combined with the probabilistic behavior QoS behavior of services, this provides an accurate portrayal of the composite service's end-to-end QoS.

Though formal analysis of end-to-end QoS has been studied in Cardoso et al. [4], there are no practical testing tools available for the composite service provider. The pairwise testing procedure has been shown to outperform other testing techniques in [5]. We extend this testing tool to develop a generic testing methodology to query end-to-end QoS of a web service. Related empirical studies of optimal QoS compositions make use of genetic programming in Canfora et al. [3] and linear programming in Zeng et al. [23]. These are dynamic techniques to choose the best possible atomic services and configurations for SLAs. The goal in our paper is to analyze the dynamic configurations that may result due to invocation/non-invocation of particular web services when atomic SLAs have already been established.

10. CONCLUSIONS

Effective computation of offline QoS tradeoff analysis is beneficial to ensure efficient portrayal of composite service behavior. In this paper, we sample large orchestration spaces with configurations satisfying pairwise interactions between services. This emanates from the intuition of pairs of service combinations lead to significant changes in composite service QoS. As demonstrated in the paper, such pairwise analysis scheme is able to reduce the number of configurations drastically, while still successfully analyzing significant ranges of QoS variance. It has been shown to be robust over both large FDs (C^3MS and eHealth) as well as over many samples of solutions. When compared to random generation of configurations, the experiments show superior performance of pairwise relating to efficiency and stability of results. To generate robust SLAs, it is beneficial to make use of systematic sampling rather than variable random sampling. It allows creation of many families of composite services that can be leveraged at varying prices and SLA ranges. Post-deployment QoS variance is minimized due to the extensive analysis of the host of orchestrations that may be constructed.

A feature model driven variability analysis of web services orchestrations can lead to useful analysis tools. An area of future work can result in *automatic generation of orchestrations* keeping feature model specialization as a causality measure. Generating these valid orchestrations and automatically synthesizing QoS behavior relieves the need for manual synthesis of orchestrations.

11. REFERENCES

[1] S. Bistarelli and F. S. Santini. Soft constraints for quality aspects in service oriented architectures. In *Fourth European Young Researchers Workshop on Service Oriented Computing, Italy*, 2009.

[2] M. Bruno, G. Canfora, M. D. Penta, G. Esposito, and V. Mazza. Using test cases as contract to ensure service compliance across releases. In *Proc. of the 3rd Intl. Conf. in Service-Oriented Computing, Amsterdam, The Netherlands*, 2005.

[3] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani. An approach for qos-aware service composition based on genetic algorithms. In *Conf. on Genetic and evolutionary computation, USA*, 2005.

[4] J. Cardoso, J. Miller, A. Sheth, and J. Arnold. Modeling quality of service for workflows and web service processes. Technical report, LSDIS Lab Technical Report, University of Georgia, 2002.

[5] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The aetg system: An approach to testing based on combinatorial design. *IEEE Trans. on Software Engineering*, 23:437–444, 1997.

[6] M. B. Cohen, M. B. Dwyer, and J. Shi. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. *IEEE Trans. on Software Engineering*, 34, 5:633–650, 2008.

[7] I. Constantinescu, B. Faltings, and W. Binder. Large scale testbed for type compatible service composition. In *ICAPS 04 workshop on planning and scheduling for web and grid services*, Whistler, Canada, June 2004.

[8] S. Y. Hwang, H. Wang, J. Tang, and J. Srivastava. A probabilistic approach to modeling and estimating the qos of web-services-based workflows. *Elsevier Information Sciences*, 177:5484–5503, 2007.

[9] M. Jaring and J. Bosch. Representing variability in software product lines: A case study. *Proc. of the Second Intl. Conf. on Software Product Lines, London, UK*, pages 15–36, 2002.

[10] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-oriented domain analysis (foda) feasibility study. *Software Engineering Institute*, 1990.

[11] A. Kattepur, S. Sen, B. Baudry, A. Benveniste, and C. Jard. Variability modeling and qos analysis of web services orchestrations. *IEEE International Conference on Web Services (ICWS)*, July 5-10, 2010 (Submitted).

[12] J. Kienzle, N. Guelfi, and S. Mustafiz. Crisis management systems: A case study for aspect-oriented modeling. Technical report, McGill Univ., 2009.

[13] D. R. Kuhn and D. D. Wallace. Software fault interactions and implications for software testing. *IEEE Trans. on Software Engineering*, 30:418–421, 2004.

[14] R. Mietzner, A. Metzger, F. Leymann, and K. Pohl. Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications. In *Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems, pp. 18-25*, 2009.

[15] J. Misra and W. R. Cook. Computation orchestration: A basis for wide-area computing. *Software and Systems Modeling, Springer*, 6(1):83–110, 2007.

[16] A. Paschke and M. Bichler. Knowledge representation concepts for automated sla management. *Journal of Decision Support Systems*, 46:187–205, 2008.

[17] M. D. Penta, G. Canfora, and G. Esposito. Search-based testing of service level agreements. In *Proc. of the 9th Conf. on Genetic and evolutionary computation, London, England*, 2007.

[18] G. Perrouin, S. Sen, J. Klein, B. Baudry, and Y. le Traon. Automatic and scalable t-wise test case generation strategies for software product lines. In *Proc. of Intl. Conf. on Software Testing*, 2010.

[19] S. Rosario, A. Benveniste, S. Haar, and C. Jard. Probabilistic qos and soft contracts for transaction-based web services orchestrations. *IEEE Trans. on Services Computing*, 1(4):187–200, 2008.

[20] SAP. Enterprise services architecture for healthcare - a prescription for innovation. *Solution Brief, Germany*, 2006.

[21] P. Schobbens, P. Heymans, J. Trigaux, and Y. Bontemps. Generic semantics of feature diagrams. *Computer Networks, Elsevier*, 51:456–479, 2007.

[22] V. Tosic and B. Pagurek. On comprehensive contractual descriptions of web services. In *IEEE Intl. Conf. on e-Technology, e-Commerce and e-Service*, pages 444–449, 2005.

[23] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. *IEEE Trans. on Software Engineering*, 30, 5:311–327, 2004.