Variability Modeling and QoS Analysis of Web Services Orchestrations

Ajay Kattepur* , Sagar Sen*, Benoit Baudry*, Albert Benveniste* and Claude Jard†
*IRISA/INRIA, Campus Universitaire de Beaulieu, 35042, Rennes-Cedex, France.
email: [Ajay.Kattepur, Sagar.Sen, Benoit.Baudry, Albert.Benveniste, Claude.Jard] @irisa.fr
†ENS Cachan, IRISA, Université Européenne de Bretagne, Bruz, France.

***Abstract*—The ever-growing choice in diverse services is making *service orchestration variability* an essential aspect of a composite web service. Influence of this variation on the Quality of Service (QoS) of a composite service is critical and the focus of our work. In this paper, we present a methodology to first model orchestration variability using a *feature diagram* (FD). The FD specifies a product line of orchestrations represented as *configurations* of invoked/rejected atomic services. Second, due to the potentially large set of configurations we employ combinatorial testing techniques to automatically generate configurations covering all valid *pairwise interactions* between services. Third, we analyze QoS variation for each configuration using probabilistic models of QoS. Using a *crisis management system* case study we experimentally show that pairwise generation covers all QoS outliers and eliminates analysis of $> 75\%$ of all possible configurations. The QoS analysis of the pairwise configurations reveals unsafe/ineffective configurations, helps determine realistic Service Level Agreements (SLAs), and provides valuable feedback to help remodel an orchestration.**

I. Introduction

Inherent choice in an ever-growing world of services is making *orchestration variability* a significant aspect of a composite web service. The different ways of orchestrating atomic services can be seen as either multiple variants of a composite service created offline or an online composite service that reconfigures dynamically. In either case, we expect to observe variation in Quality of Service (QoS) across different orchestrations. This variation in QoS must not only take into account service variability but also the uncertainty/probabilistic nature of QoS itself.

It is important to consider orchestration variability and its implications on composite service behavior. For instance, not considering variability leads to misrepresentation of contractual agreements on QoS [1]. Contractual agreements such as service level agreements (SLAs) [2] is the industry standard to ensure QoS compliance between service providers and customers. Usual deviations from SLAs are a result of non-incorporation of QoS variability and in particular QoS outliers in its specification. Therefore, we need systematic analysis of variability in order to improve robustness of contractual SLAs.

Modeling variability in web service orchestrations and analyzing the consequent variation in QoS is the principal subject of this paper. We present a methodology to model orchestration variability using *feature diagrams* (FDs). Feature diagrams [3] provide a graphical constraints-based framework to specify a product-line of orchestrations. Each orchestration in the product-line is represented as an authorized configuration of invoked/rejected atomic services. In most cases the FD specifies a very large set of configurations making exhaustive sampling infeasible. Instead, we sample the set of all possible configurations by systematically analyzing configurations covering all valid pairwise service interactions [4]. Finally, we use probabilistic models of QoS [5] to analyze variants of orchestrations derived from all valid configurations.

We use our methodology to investigate merits of systematically sampling the set of all configurations of web service orchestrations. 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. Moreover, random sampling is not easy when FD constraints like mutual exclusion/requirement need to be satisfied. This work focuses on the adaptation of combinatorial interaction testing (CIT) [6] to select a sample of configurations that covers all pairwise interactions of services while satisfying all FD constraints. We use the recently proposed scalable approach in [7] for generating these configurations. CIT is based on the observation that most of the faults are triggered by interactions between a small number of variables [8]. For example, consider the output quality of printing web pages depending on a hypothetical combination of parameters represented in Table I. An exhaustive generation of combinations of these parameter options would entail 1536 cases with many redundancies. Pairwise coverage of optional combinations would require just 17 tests, resulting in a reduction of close to 99%. The number of exhaustive tests will increase exponentially with addition of more parameters/options requiring an employment of efficient sampling strategies.

Parameters	**Options**
Operating System	Windows, Linux, Macintosh
Browser	IE, Firefox, Chrome, Opera
Printer Model	HP, Canon, Xerox, Epson
Printer Type	Ink-Jet, Laser
Orientation	Portrait, Landscape
Size	A3, A4, A5, A6
Color	B/W, Multicolor

TABLE I
Examples of printing parameters requiring comparison.

Pairwise coverage test generation has been used to detect faults in software systems in prior work [4], [6]. However, the application of these coverage-based techniques to sample configurations in service orchestrations is yet to be examined. This work performs such an examination through a series of experiments that aim at investigating several facets of the question: is pairwise service interaction sampling of orchestration configurations effective for overall QoS analysis and the consequent definition of a global SLA?

All experiments are based on a *crisis management system*

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).

(CMS) case study described comprehensively in [9]. This paper reports on the following questions:

- Is it possible to automatically sample the orchestration configurations space to select configurations that cover all pairwise service interactions?
- What global QoS metrics can we infer from a pairwise sample?
- How stable is the SLA computed from a pairwise sample? This question is related to the fact that the automatic generation of pairwise configurations is not deterministic and thus the global contract might vary depending on the generated *sample*.
- Is pairwise sampling more effective and efficient compared to exhaustive 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 small set of configurations satisfying pairwise interactions. Consistency of various generated pairwise solutions are also demonstrated through simulations. This comprehensive analysis of variability helps the orchestrator understand the global QoS extremities of the composite service before negotiating a SLA agreement. Deterioration in service quality or non-compliance of SLA standards during online deployment of the service is thus prevented. Improvements in the orchestration model to eliminate some deviant configurations (causing excessive deterioration of end-to-end QoS) or grouping a family of configurations with similar QoS behavior are other extensions of this technique.

This paper is organized as follows. Section II provide foundations required for our methodology. These include feature diagrams in II-A, Orc language for orchestration in II-B, pairwise configuration generation in II-C and formal description of QoS metrics in II-D. The methodology followed in this paper is briefly presented in Section III. In Section IV the crisis management system (CMS) is described. Comprehensive analysis of the CMS case study is done in Section V. Emphasis was placed on the probabilistic distribution simulations in V-A and efficient pairwise generation of configurations in V-B. Evaluation of these schemes to generate families of QoS output was done in V-C. Study of the robustness of pairwise interactions and its comparison with exhaustive configurations was also done in V-D. Related work in literature is presented in Section VI followed by conclusions and perspectives in Section VII.

II. Foundations

A. Modeling Variability in Composite Services

Variability in a composite service derives from choice in several available online services. Each of these configurations represents a set of invoked or rejected atomic services. Selection of some services in a configuration may compulsorily link the selection of other services. Some services are mutually exclusive and cannot be selected simultaneously. In this paper, we model the variability in service configurations using a feature diagram (used interchangeably with feature model) often used to model Software Product Lines (SPLs).

Feature Diagrams (FD) introduced by Kang et al. [3] compactly represent all the products of a SPL (referred to as *configurations* in this paper) in terms of features which can be composed. Feature diagrams have been formalized to perform SPL analysis [10]. In [10], 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 $f_1, f_2, ..., f_k$
- A 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 further children is called a leaf.
- A parent-child relationship between features f_p and f_c are categorized as follows:
  - *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.
  - *Alternative (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 invocations/rejections.

B. Service Orchestration using Orc

While the FD describes a set of services invoked/rejected, it is crucial to formally describe the causal link between the invoked atomic services using an orchestration. The business process execution language (BPEL) [11] has been used as the industry standard for describing orchestrations. However, the use of this language has certain disadvantages including inherent complexity of the language and restrictions in combinatorial service descriptions [12]. Orc [13] serves as a simple yet powerful concurrent programming language to describe web services orchestrations.

The fundamental declaration used in the Orc language is a *site*. Sites can be both *external* or *internal*. When an external *site* is made available to Orc, its type is also made available to the Orc. 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 [13]. 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, the expression $F|G$ 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 in G. 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.

C. Configuration Generation from Feature Diagram

Combinatorial interaction testing (CIT) has been proposed by Cohen et al. [6] 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-wise 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-wise testing which samples the input domain to cover all t-wise combinations.

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

In this definition, N is the number of experiments, the strength t of the array is the parameter that allows achieving 2-wise (pairwise), 3-wise or t-wise 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 [6]. 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. 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 [7]. 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 as *samples* from now) 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.

D. 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 [5] [14] provides a more realistic study of actual web services' behavior.

The following QoS parameters have been chosen:

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 II 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 [15] 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 II
QoS METRICS DISCUSSED IN [15] EXTENDED TO ORC COMBINATORS.

III. METHODOLOGY

We present a methodology designed to examine: (a) A superior technique for sampling the possible configurations to ensure efficient portrayal of QoS behavior of a composite service; (b) The need for probabilistic analysis of QoS in variable service orchestrations. The following steps summarize our methodology:

1) The inputs are: (a) Variability and constraints of a set of configurations of services modeled in a FD; (b) A composite service orchestration in Orc to specify causality and service interactions. The modeling inputs may be specified as a 3-tuple (S, FD, O) where:
   - S is the set of services that can be used. In a configuration, subsets $S_1, ..., S_N$ of these services are used.
   - FD is the constraints for the services included in a particular configuration.
   - 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.
3) For each of the sampled configurations we analyze the QoS for orchestrations invoking all atomic services in the configuration. 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 along with comparison with other QoS metrics.
4) Comparisons with randomly generated configurations and consistency over multiple sample sets is included to experimentally study the robustness of the proposed pairwise analysis scheme.

For the rest of the paper, we explain in detail this methodology applied to the crisis management system case study.

IV. Crisis Management System Case Study

Drawing from the comprehensive documentation in [9], the chosen composite service models a typical crisis management system (CMS). The need for such crisis management systems has grown significantly over time with efficient collaboration of various (distributed) parties responsible for speedy assistance and recovery. A crisis can range over major catastrophes like natural disasters, terrorist attacks, accidents and technological disruptions. These are examples of emergency situations that are unpredictable and lead to severe after-effects unless handled immediately. A CMS facilitates this process by orchestrating the communication, co-ordination and deployment between all parties involved in handling the crisis. A thorough analysis of QoS aspects of a CMS will not only ensure optimal performance of such mission critical systems, but also ensure speedy and reliable assistance to the parties in need of aid.

A. Feature Diagram of CMS

In Figure 1, we present the Crisis Management System (CMS) FD [9]. The CMS FD contains several features that are associated with software assets represented by atomic services. For example, the *Local Operator* feature is represented by the *GSMLocalOperator* web service. Constraints such as optional, requires and mutual exclusion (XOR) are also incorporated. For example, the *LocalOperator* and *InternationalOperator* features are mutually exclusive while the *HospitalAdmit* feature requires the *Ambulance* feature.

B. Service Orchestrations in CMS

A host of web services used for the orchestration are described in detail in Table III.

The FD (Fig. 1) and the orchestration (Fig. 2) 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 [10], *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 [10] to ensure compatibility of an orchestration with the feature model as follows:

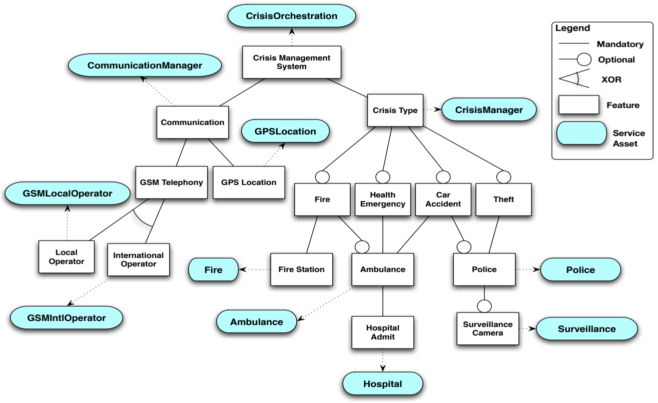

Fig. 1. Feature Diagram / Model of the Crisis Management System with associated real-world service assets.

Web Service	Description
CrisisOrchestration	Uses the customer input to orchestrate the CMS system
CrisisManager	Selects the emergency services to include in the orchestration
CommunicationManager	Selects the communication services to include in the orchestration
GPSLocation	Sets up the GPS location of the emergency area
GSMLocalOperator	Sets up a local GSM communication link for personnel
GSMIntlOperator	Sets up an international GSM communication link for personnel
Ambulance	Contacts and waits for a response from nearby ambulance agencies
Hospital	Contacts and waits for a response from nearby hospitals
Police	Contacts and waits for a response from nearby police stations
Surveillance	Connects to surveillance tapes from the affected area
Fire	Contacts and waits for a response from fire stations

TABLE III
Web Services in the CMS Orchestration.

- 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 Fig. 1, the composite service may be developed by an orchestrator. Automatic compositions of composite services from feature model constraints (with additional attributes to describe orchestration interactions), is out of the scope of this paper and will be investigated in future work.

The composite service orchestration is represented succinctly in Fig. 2 and the Orc representation is presented in Table IV. Calling the *CrisisOrchestration* service invokes the *CommunicationManager* and *CrisisManager* operations in sequence. The *CommunicationManager* service calls the *GPSLocation* and either one of the *GSMLocalOperator* and the *GSMIntlOperator* services that are mutually exclusive (*Mux*). The outputs are synchronized and merged (*Merge*) before dynamically invoking the optional services through the *CrisisManager*. The varying timer values are used to invoke / discard the *Fire*, *Ambulance*, *Hospital*, *Police* and

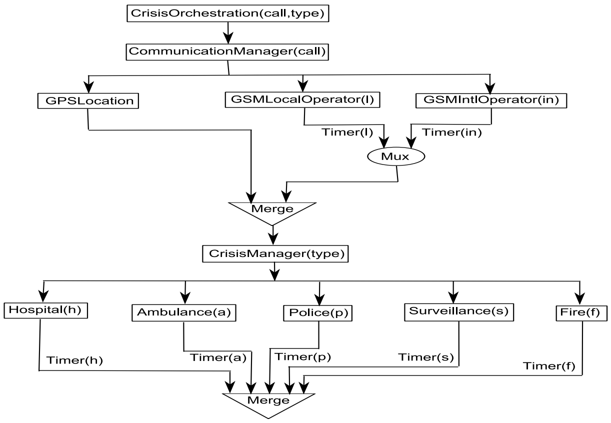

Fig. 2. Composite Web Service Orchestration of the CMS.

CrisisOrchestration (call,type) $\triangleq$ *CommunicationManager(call)* $\gg$ *CrisisManager(type)*
CommunicationManager(call) $\triangleq$ *(l,in)* $\gg$ *(LocalOperator(in),IntlOperator(in),GPSLocation())*
GPSLocation() $\triangleq$ *(x,y)*
GSMLocalOperator(l) $\triangleq$ let(*query(l)* $|$ *Timer(l))*
GSMIntlOperator(in) $\triangleq$ let(*query(in)* $|$ *Timer(in))*
CrisisManager(type) $\triangleq$ *(f,a,h,p,s)* $\gg$ *(Fire(f),Ambulance(a),Hospital(h),Police(p),Surveillance(s))*
Fire(f) $\triangleq$ let(*query(f)* $|$ *Timer(f))*
Ambulance(a) $\triangleq$ let(*query(a)* $|$ *Timer(a))*
Hospital(h) $\triangleq$ let(*query(h)* $|$ *Timer(h))*
Police(p) $\triangleq$ let(*query(p)* $|$ *Timer(p))*
Surveillance(s) $\triangleq$ let(*query(s)* $|$ *Timer(s))*

TABLE IV
ORC REPRESENTATION OF THE CMS ORCHESTRATION.

Surveillance services. The outputs of these services are merged and synchronized. In the Orc model presented in Table IV, the generic service *query()* is used to represent the invocation of a particular web service. The setting of timer values (*Timer()*) results in the various associated configurations in the system and is an example of defining orchestration parameters. 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.

V. EXPERIMENTS

We perform experiments using the methodology described in Section III for the CMS case study. This involved simulating probabilistic QoS of atomic services, pairwise generation of configurations and finally, analysis of composite services' probabilistic QoS behavior for the variable configurations.

A. Simulation of QoS Distributions

The first step is simulating the probabilistic response time distributions of each atomic web service as done in [5]. For this, we make use of the *t-location distribution* fitting feature in MATLAB as shown in Fig. 3. 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 to generate various distributions of services' response times with varying parameters. An example of this for 10,000 runs is as shown in Fig. 4.

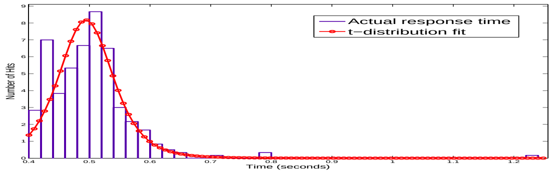

Fig. 3. Distribution fitting of actual response times of a web service invocation.

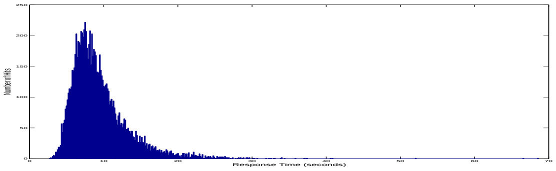

Fig. 4. Response time distributions of services with $\nu = 5$ and $\delta = 8$ seconds.

B. Generating a sample of configurations for CMS

We transform the CMS FD to constraint satisfaction problem model in the language Alloy as described in [7]. 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.

A set of 15 configurations, $\mathbf{C1}$ to $\mathbf{C15}$, were deemed sufficient by the pairwise generation methodology to represent the configuration sample space. These are shown in Table V with a $\times$ representing service invocation. Guidelines for setting experimental parameters in order to efficiently generate solutions may be found in [7].

In conjunction to the configurations in Table V examples of two generated cases are shown in Fig. 5. These configurations cover tuples specified in nos. **C1** and **C15**. While the basic configuration **C15** has none of the optional services, **C1** has three of the optional services invoked. Such variability in orchestration can produce radically different QoS values.

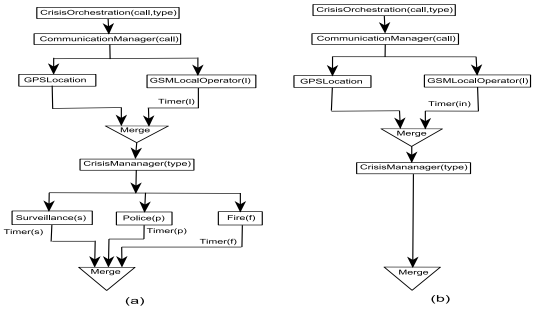

Fig. 5. Varying configurations of the atomic services (a) Configuration **C1** (b) Configuration **C15**.

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 configurations in a FD by a self-reconfiguring composite service. This would create orchestrations dynamically and link them to a particular

Web Service	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**	**C9**	**C10**	**C11**	**C12**	**C13**	**C14**	**C15**
CrisisOrchestration	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
CommunicationManager	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
CrisisManager	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
GPSLocation	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
GSMLocalOperator	×	×	×				×	×				×			×
GSMIntlOperator				×	×	×			×	×	×		×	×	
Fire	×			×			×		×			×	×	×	
Ambulance		×	×	×		×		×		×		×	×	×	
Hospital		×	×	×		×		×		×		×	×	×	
Police	×	×		×				×	×	×	×	×	×		
Surveillance	×	×		×				×			×	×			

TABLE V

WEB SERVICES IN THE ORCHESTRATION AND THE VARIABLE CONFIGURATIONS (**C1** TO **C15**) WITH × REPRESENTING A SERVICE INVOCATION.

FD configuration. However, due to the added control of systematic configuration generation from FDs, we resort to static invocation of orchestrations.

C. Evaluating QoS of a Composite Service

The efficacy of the QoS analysis procedure was tested experimentally. The web services of the CMS were assigned random response times from a range of heterogeneous t-distributions. The range of parameter values for these distributions in MATLAB included degrees of freedom (ν) varying from 3 to 6 and non-centrality (δ) varying from 5 to 10 seconds.

For an invoked service, the individual timeout value was set sufficiently high (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 performed. The response time of the orchestration was collected during each run to generate an associated distribution.

As seen in Fig. 6, the pairwise generated configurations cover a range of response time distributions. The three worst performing configurations (**C4**, **C8**, **C12**) are compared as an example. The median and 90 percentile changes between these configurations are shown. This demonstrates the use of a few configurations to test significant changes in QoS parameters in a composite service.

In Fig. 6, the three worst performing configurations have a significant contribution to the percentile deviations of the response time distribution. This is further seen in the *box-plot* representation in Fig. 7. On each box, the *red* 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. The three worst performing configurations (**C4**, **C8**, **C12**), in terms of response times' values, are once again compared in the box-plot (horizontal dotted lines passing through the medians).

Additional parameters such as availability of a service, the cost entailed in calling atomic services and output data quality is also studied in tandem. Using the combinators described in Table II, the QoS parameters were analyzed for each configuration generated by the pairwise interactions. Setting atomic service availability to $\mathbf{0.95}$ (representing service availability in 95% of invocations) the composite availability each configuration is shown in Table VI. 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). For example, setting the $\chi = 5$ units for each invoked atomic service, the cost of each configuration is shown in Table VI. Furthermore, setting $\kappa = 20$, the output data quality of the configurations may also be derived. A higher availability and data quality with lower costs and response times are desirable. For example, comparing **C3** and **C4**, calling additional services entails lower availability and higher costs to the orchestrator, albeit with additional output data quality. Though simplistic in outlook (due to subjectivity of cost and data quality of atomic services), this trade-off of parameters must be taken into account. These myriad of QoS parameters accurately quantify run-time behavior of the composite service.

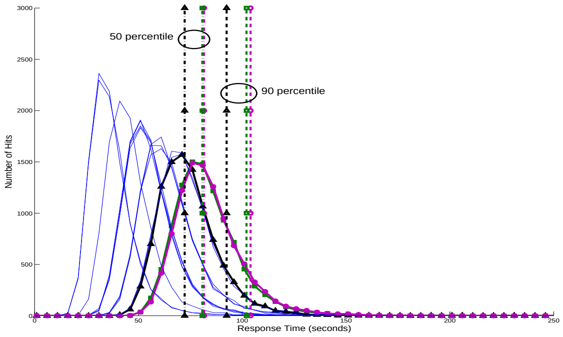

Fig. 6. Response times of the pairwise configurations with emphasis on comparing the three configurations with highest response times.

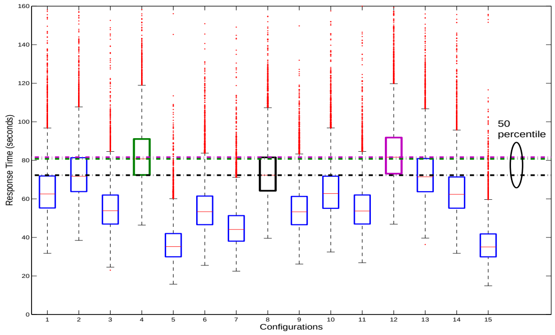

Fig. 7. Box-plot representation of the pairwise configurations with the median values marked for the extreme cases.

From these results, the orchestrator can have a global overview of the performance of the composite service. The possibilities include:

1) Setting the SLA keeping into account the worst performing configuration. This will prevent contract deviation during actual deployment of the service.
2) Setting a family of SLAs for a set of configurations taking

Metric	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**	**C9**	**C10**	**C11**	**C12**	**C13**	**C14**	**C15**
Availability (α)	0.6634	0.6302	0.6983	0.5987	0.7738	0.6983	0.7351	0.6302	0.6983	0.6634	0.6983	0.5987	0.6302	0.6634	0.7738
Cost (χ)	40	45	35	50	25	35	30	45	35	40	35	50	45	40	25
Data Quality (ξ)	2.0000	2.2500	1.7500	2.5000	1.2500	1.7500	1.5000	2.2500	1.7500	2.0000	1.7500	2.5000	2.2500	2.0000	1.2500

TABLE VI
AVAILABILITY, DATA QUALITY AND COST OF THE PAIRWISE CONFIGURATIONS.

into account trade-offs between QoS metrics and the output quality of configurations. This leads to a product line of composite services with extensively analyzed SLAs. For example, the configurations **C2**, **C8** and **C13** with very similar characteristics can be grouped as a separate line of services.

3) Eliminating certain deviating configurations to improve the overall performance. This may be done by adding further constraints in the orchestration/feature models. For example, consider the services **C4** and **C12**. Eliminating these configurations (by addition of constraints) reduces the output data quality by 0.25 units as seen Table VI. However, it improves the 90, 50, 75 and 25 percentiles of the overall response time distributions by $\mathbf{11.53}$, $\mathbf{10.3}$, $\mathbf{9.3}$ and $\mathbf{8.87}$ seconds respectively. These are significant durations if the orchestrator of a composite service is vying to compete with other companies offering lower response time durations for similar quality services.

Using the pairwise analysis scheme, these imperative qualitative results are obtained with quantitative efficiency even when the number of services are considerably large.

D. Evaluating the Pairwise Sampling Technique

To experimentally test the efficacy of combinatorial testing the $\mathbf{15}$ pairwise configurations (Table V) were compared with all the $\mathbf{64}$ exhaustive independent configurations of the CMS orchestration. As shown in Fig. 8, the comparison is made using the 25, 50, 75 and 90 percentiles of response time distributions for 10,000 Monte-Carlo runs in MATLAB. These families of exhaustive configurations (with few millisecond redundant deviations) are represented by one pairwise configuration. The pairwise configurations are able to capture the extreme values representing greater than $\mathbf{55}$ seconds of quantile deviation. This represents greater than 75% decrease in the number of exhaustive tests, which will increase in an exponential fashion with introduction of new services. The accuracy of the pairwise sampling scheme is further demonstrated in Table VII where the *mean* and *maximum* deviations of the pairwise values from the nearest exhaustive values are provided. These are expressed as a percentage of the mean inter-family response time difference (8.96 seconds). Thus, for such orchestrations with numerous configurations, using pairwise interactions is a sufficient choice in order to examine the entire sample space. 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 $\mathbf{40}$ samples that satisfy the pairwise interaction testing were generated. The statistics of the worst performing configuration (with highest response time) in each sample was collected through 10,000 Monte-Carlo runs and is shown in Fig. 9. These are consistent overall with the highest variance of 0.8 seconds seen in the 90 percentile value. This may be attributed to outliers included in the extreme configurations. Thus, it may be essential to analyze a range of percentiles to accurately estimate the deviation of particular configurations. Use of more than one sample should improve robustness of the offline analysis framework as certain extreme configurations may not occur always. Use of domain specific information may also be required to further ensure robustness of samples.

Percentile values	25	50	75	90
Mean	1.1326%	1.3471%	1.3438%	1.5471%
Maximum	9.0075%	7.2147%	7.1243%	5.2030%

TABLE VII
DEVIATIONS OF THE PAIRWISE AND EXHAUSTIVE ANALYSIS VALUES.

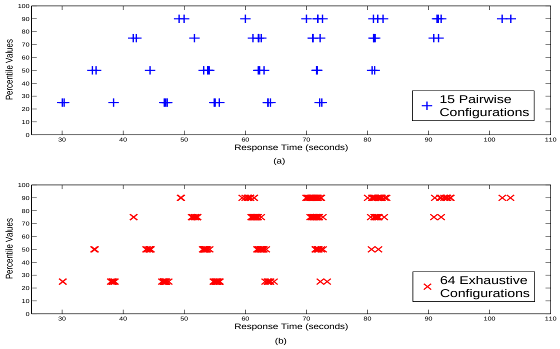

Fig. 8. Comparison of pairwise and exhaustive generation of configurations with 25, 50, 75 and 90 percentile values of response time distributions analyzed.

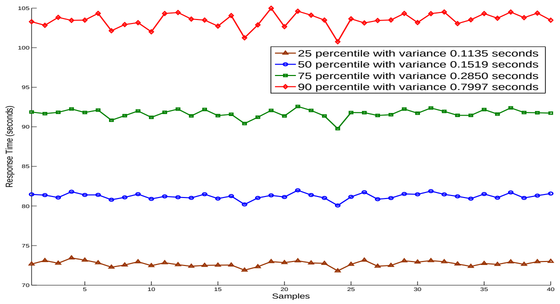

Fig. 9. Percentile values of most deviant scenarios generated by pairwise interactions for the CMS orchestration.

VI. RELATED WORK

The combinatorial testing framework described by Cohen et al. [6] has been applied extensively to efficient testing for fault detection. In the work of Cohen et al. [17], 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 [16] 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 solver proposed by Perrouin et al. [7] is used. In [18] Larsen et al. define modal I/O automata, an extension of interface automata with modality. These allow models of varying configurations to be developed from a single produce line while disallowing trivial implementations. Such a notion when extended to a composite service can provide interesting configurations and versions of composite products as described in [18].

Pre-deployment testing of SLAs has been studied by Di Penta et al. [19], 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 [20], 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 [5] and Bistarelli et al. [21]. 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. [14] 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. The hard contract notions of end-to-end QoS are replaced by the probability quantile based approach. This provides the service provider the technique for estimating composite service QoS distributions and estimating the global soft contract SLA. Though formal analysis of end-to-end QoS has been studied in Cardoso et al. [15], there are no practical testing tools available for the service provider. The pairwise testing procedure has been shown to outperform other testing techniques in [6]. We extend this testing tool to develop a generic testing methodology to query end-to-end QoS of a web service. The efficacy of this scheme is provided though experimental verification.

Related empirical studies of optimal QoS compositions make use of genetic programming in Canfora et al. [22] and linear programming in Zeng et al. [23]. These are dynamic techniques to choose the best possible atomic services and configurations keeping QoS in mind. This differs from our work as they assume that there are choices in the best possible atomic web services. The goal in our paper is to analyze the variable configurations that may result due to invocation or non-invocation of particular web services.

VII. Conclusion and Perspectives

Accurate offline analysis of a composite web service before its deployment is essential to ensure non-repudiation of a SLA contract. This is necessary to maintain optimal QoS behavior of mission-critical services such as crisis management. In order to do this, the service provider must keep in mind the probabilistic aspect of QoS parameters and the variable configurations in a composite service. In this paper, we study an analysis framework to test the QoS of an orchestration before deployment. Further, the notion of systematic pairwise sampling procedure has also been demonstrated, which provides a more efficient sampling of the configuration space than exhaustive trails while still maintaining sufficient coverage. Larger FD and orchestration models can be analyzed using the divide-and-compose approaches [7] to handle this scalability issue. This should provide a simple, systematic and stochastically correct methodology for pre-deployment QoS analysis of a composite service.

While this paper concentrates on a particular composition of fixed atomic services, a future area of interest would be optimal compositions. The use of configurations and scenarios modeled by a FD leads to a family of composite services. These, in turn, may be used to generate many versions of the orchestrations. Further implementation of these techniques to study larger composite orchestrations is useful for both obtaining realistic QoS bounds and product generation of families of services.

References

[1] V. Tosic and B. Pagurek, “On comprehensive contractual descriptions of Web services,” *IEEE Intl. Conf. on e-Technology, e-Commerce and e-Service*, pp. 444–449, 2005.

[2] A. Paschke, and M. Bichler, “Knowledge Representation Concepts for Automated SLA Management,” *Journal of Decision Support Systems*, vol. 46, pp. 187–205, 2008.

[3] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson, “Feature-Oriented Domain Analysis (FODA) Feasibility Study,” *Software Engineering Institute*, 1990.

[4] J. Czerwonka, “Pairwise Testing in the Real World,” *24th Pacific Northwest Software Quality Conference*, 2006.

[5] 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*, vol. 1, no. 4, pp. 187 – 200, 2008.

[6] 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*, vol. 23, no. 7, pp. 437–444, 1997.

[7] 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,” *Proc. of Intl. Conf. on Software Testing*, April 2010.

[8] D. R. Kuhn, and D. D. Wallace, “Software Fault Interactions and Implications for Software Testing,” *IEEE Trans. on Software Engineering*, vol. 30, no. 6, pp. 418–421, 2004.

[9] J. Kienzle, N. Guelfi, and S. Mustafiz, “Crisis Management Systems: A Case Study for Aspect-Oriented Modeling,” *McGill Univ.*, Technical Report, 2009.

[10] P. Schobbens, P. Heymans, J. Trigaux, and Y. Bontemps, “Generic semantics of feature diagrams,” *Computer Networks*, Elsevier, vol. 51, pp. 456–479, 2007.

[11] IBM, BEA Systems, Microsoft, SAP AG, and Siebel Systems, “Business Process Execution Language for Web Services,” version 1.1, 2007.

[12] P. M. Kelly, P. D. Coddington, and A. L. Wendelborn, “A Simplified Approach to Web Service Development,” *Australasian workshops on Grid computing and e-research*, vol. 54, pp. 79–88, 2006.

[13] J. Misra and W. R. Cook, “Computation Orchestration: A Basis for Wide-area Computing,” *Springer J. of Software and Systems Modeling*, vol. 6, no. 1, pp. 83 – 110, Mar. 2007.

[14] 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*, vol. 177, pp. 5484–5503, 2007.

[15] J. Cardoso, J. Miller, A. Sheth, and J. Arnold, “Modeling Quality of Service for Workflows and Web Service Processes,” *LSDIS Lab Technical Report*, University of Georgia, pp. 1–44, 2002.

[16] 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, pp. 15–36, 2002.

[17] 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*, vol. 34, no. 5, pp. 633–650, 2008.

[18] K. G. Larsen, U. Nyman, and A. Wasowski, “Modal I/O automata for interface and product line theories,” *Joint European Conf. on Theory and Practices of Software*, Braga, Portugal, pp. 64–79, 2007.

[19] M. Di Penta, G. Canfora, and G. Esposito, “Search-based Testing of Service Level Agreements,” *Proc. of the 9th Conf. on Genetic and evolutionary computation*, London, England, pp. 1090–1097, 2007.

[20] M. Bruno, G. Canfora, M. Di Penta, G. Esposito, and V. Mazza, “Using Test Cases as Contract to Ensure Service Compliance Across Releases,” *Proc. of the 3rd Intl. Conf. in Service-Oriented Computing*, Amsterdam, The Netherlands, pp. 87–100, 2005.

[21] S. Bistarelli and F. S. Santini, “Soft Constraints for Quality Aspects in Service Oriented Architectures,” *Fourth European Young Researchers Workshop on Service Oriented Computing*, Italy, 2009.

[22] G. Canfora,M. Di Penta, R. Esposito, and M. L. Villani, “An approach for QoS-aware service composition based on genetic algorithms,” *Conf. on Genetic and evolutionary computation*, USA, pp. 1069–1075, 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*, vol. 30, no. 5, pp. 311–327, 2004.