QoS-aware Complex Event Service Composition and Optimization using Genetic Algorithms *

Feng Gao[1], Edward Curry[1], Muhammad Intizar Ali[1], Sami Bhiri[2], and Alessandra Mileo[1]

[1] INSIGHT Centre,
NUI, Gawaly, Ireland.
firstname.lastname@insight-centre.org
[2] Computer Science Department,
TELECOM SudParis, France.
sami.bhiri@telecom-sudparis.eu

Abstract. The proliferation of sensor devices and services along with the advances in event processing brings many new opportunities as well as challenges. It is now possible to provide, analyze and react upon real-time, complex events about physical or social environments. When existing event services do not provide such complex events directly, an event service composition maybe required. However, it is difficult to determine which compositions best suit users' quality-of-service requirements. In this paper, we address this issue by first providing a quality-of-service aggregation schema for event service compositions and then developing a genetic algorithm to efficiently create optimal compositions.

Keywords: event service composition, genetic algorithm, quality-of-service

1 Introduction

Recent developments in the Internet-of-Things (IoT) services envision "Smart Cities", which promise in improving urban performances in terms of sustainability, high quality of life and wiser management of natural resources. Complex Event Processing (CEP) and event-based systems are important enabling technologies for smart cities [4], due to the need for integrating and processing high volumes of real-time physical and social events. However, with the multitude of heterogeneous event sources to be discovered and integrated [3], it is crucial to determine which event services should be used and how to compose them to match non-functional requirements from users or applications. Indeed, non-functional properties, e.g.: quality-of-service (QoS) properties, can play a pivotal role in service composition [7].

In this paper, we extend the work in [2], which aims to provide CEP applications as reusable services and the reusability of those event services is determined

* This research has been partially supported by Science Foundation Ireland (SFI) under grant No. SFI/12/RC/2289 and EU FP7 CityPulse Project under grant No.603095. http://www.ict-citypulse.eu

by examining complex event patterns and primitive event types. This paper aims to enable a QoS-aware event service composition and optimization. In order to facilitate QoS-aware complex event service composition, two issues should be considered: QoS aggregation and composition efficiency. The QoS aggregation for a complex event service relies on how its member events are correlated. The aggregation rules are inherently different to conventional web services. Efficiency becomes an issue when the complex event consists of many primitive events, and each primitive event detection task can be achieved by multiple event services. This paper addresses both issues by: 1) creating QoS aggregation rules and utility functions to estimate and assess QoS for event service compositions, and 2) enabling efficient event service compositions and optimization with regard to QoS constraints and preferences based on Genetic Algorithms.

The remainder of the paper is organized as follows: Section 2 discusses related works in QoS-aware service planning; Section 3 presents the QoS model we use and the QoS aggregation rules we define; Section 4 presents the heuristic algorithm we use to achieve global optimization for event service compositions based on Genetic Algorithms (GA); Section 5 evaluates the proposed approach; conclusions and future work are discussed in Section 6.

2 Related Work

The first step of solving the QoS-aware service composition problem is to define a QoS model, a set of QoS aggregation rules and a utility function. Existing works have discussed these topics extensively [5, 7]. In this paper we extract typical QoS properties from existing works and define a similar utility function based on *Simple Additive Weighting* (SAW). However, the aggregation rules in existing works focus on conventional web services rather than complex event services, which has a different QoS aggregation schema. For example, event engines also has an impact on QoS aggregation, which is not considered in conventional service QoS aggregation. Also, the aggregation rules for some QoS properties based on event composition patterns are different to those based on workflow patterns (as in [5]), which we will explain in details in Section 3.1.

As a second step, different concrete service compositions are created and compared with regard to their QoS utilities to determine the optimal choice. To achieve this efficiently, various GA-based approaches are developed [8, 1, 6]. The above GA-based approaches can only evaluate service composition plans with fixed sets of service tasks (abstract services) and cannot evaluate composition plans which are semantically equivalent, but consist of different service tasks, i.e., service tasks on different granularity levels. A more recent work in [7] addresses this issue by developing a GA based on Generalized Component Services. Results in [7] indicate that up to a 10% utility enhancement can be obtained by expanding the search space. Composing events on different granularity levels is also a desired feature for complex event service composition. However, [7] only caters for *Input, Output, Precondition* and *Effect* based service composi-

Table 1: QoS aggregation rules based on composition patterns

Dimensions	**Root Node Event Operators**			
	Repetition	Sequence	And	Or
$P(\mathscr{E}), E(\mathscr{E}) =$	$\sum P(e), \sum E(e)$, where $e \in \mathscr{E}_{ice}$			
$Ava(\mathscr{E}) =$	$\prod Ava(e)$, where $e \in \mathscr{E}_{ice}$			
$S(\mathscr{E}) =$	$min\{S(e), e \in \mathscr{E}_{ice}\}$			
$L(\mathscr{E}) =$	$L(e), e$ is the last event in $\mathscr{E}_{dst}$		$avg\{L(e), e \in \mathscr{E}_{dst}\}$	
$C(\mathscr{E}) =$	$\frac{min\{C(e) \cdot f(e), e \in \mathscr{E}_{dst}\}}{card(\mathscr{E}) \cdot f(\mathscr{E})}$		$\frac{max\{C(e) \cdot f(e), e \in \mathscr{E}_{dst}\}}{f(\mathscr{E})}$	
$Acc(\mathscr{E}) =$	$\frac{card(\mathscr{E}) \cdot f(\mathscr{E})}{min\{Acc(e)^{-1} \cdot f(e), e \in \mathscr{E}_{dst}\}}$		$\frac{f(\mathscr{E})}{max\{Acc(e)^{-1} \cdot f(e), e \in \mathscr{E}_{dst}\}}$	

tions. Complex event service composition requires an *event pattern* based reuse mechanism [2].

3 QoS Model and Aggregation Schema

In this section, a QoS aggregation schema is presented to estimate the QoS properties for event service composition. A utility function is introduced to evaluate the QoS utility under constraints and preferences.

3.1 QoS Aggregation

In this paper, some typical QoS attributes are investigated, including: latency, price, bandwidth consumption, availability, completeness, accuracy and security. A numerical quality vector $Q =< L, P, E, B, Ava, C, Acc, S >$ is used to specify the QoS measures of an event service with regard to these dimensions. The *Composition Plan* is a key factor in aggregating quality vectors for event service compositions. As in [2], a composition plan contains an event pattern correlating event services with event operators. Event patterns are modeled as event syntax trees. In this paper, a step-wise transformation of event syntax tree is adopted to aggregate QoS properties. Aggregation rules for different QoS dimensions can be event operator dependent or independent, as shown in Table 1. In Table 1, $\mathscr{E}$ denotes an event service composition. $P(\mathscr{E}), E(\mathscr{E})$ etc. denote QoS values of $\mathscr{E}$. $\mathscr{E}_{ice}$ and $\mathscr{E}_{dst}$ denotes the set of Immediately Composed Event services and Direct Sub-Trees the syntax tree of $\mathscr{E}$, respectively. $f(\mathscr{E})$ gives the frequency of $\mathscr{E}$, $card(\mathscr{E})$ gives the repetition cardinality of the root node in $\mathscr{E}$.

3.2 Event QoS Utility Function

Given a quality vector Q representing the QoS capability of an event service (composition), we denote $q \in Q$ as a quality value in the vector, $O(q)$ as the theoretical optimum of q, $C(q)$ as the user-defined hard constraint on q and $0 \leq W(q) \leq 1$ as the weight of q representing users' preferences. Further, we

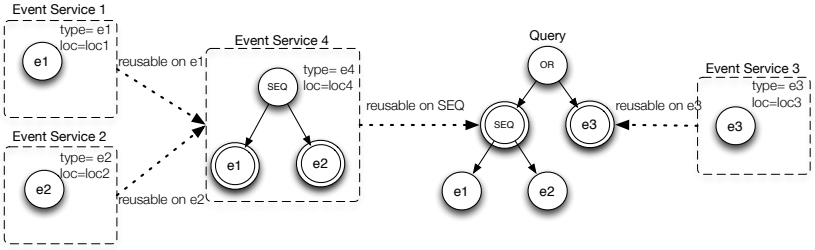


Fig. 1: Marking the reusable nodes

distinguish between QoS properties with the positive or negative tendency: $Q = Q_+ \cup Q_-$, where $Q_+ = \{Ava, R, Acc, S\}$ is the set of properties with the positive tendency (bigger values the better), and $Q_- = \{L, P, E, B\}$ is the properties with the negative tendency (smaller values the better). QoS utility U is given by:

$$U = \sum \frac{W(q_i) \cdot (q_i - C(q_i))}{O(q_i) - C(q_i)} - \sum \frac{W(q_j) \cdot (q_j - O(q_j))}{C(q_j) - O(q_j)} \quad (1)$$

where $q_i \in Q_+, q_j \in Q_-$. U should be maximized for the best candidate.

4 Genetic Algorithm for QoS-Aware Event Service Composition Optimization

We propose a heuristic method based on *Genetic Algorithms* (GA) to derive global optimizations for event service compositions, In our approach, the "fitness" of each solution can be evaluated by the QoS utility function in Equation (1). Compared to traditional GA-based optimizations for service compositions (where a composite service is accomplished by a fixed set of service tasks), event service compositions can have variable sets of sub-event detection tasks. Determining which event services are reusable to the event service request is resolved with hierarchies of reusable event services, called an Event Reusability Forest (ERF) [2]. In this section we elaborate how we utilize the ERF in the genetic algorithm for optimizing event service composition.

4.1 Population Initialization

Given an complex event service request expressed as a *canonical* event pattern *ep*, we consider the initialization of the population consists of three steps. First, enumerate all *Abstract Composition Plans* (ACPs) of *ep*. An ACP is a composition plan without concrete service bindings. Second, pick randomly a set of ACPs. Third, for each chosen ACP, pick randomly one concrete event service binding for each sub event involved. Then, a set of random *Concrete Composition Plans* (CCPs) is obtained. To create ACPs, we mark the *reusable nodes* of the event patterns in ERF. A reusable node is denoted N_r: $N_r \subseteq f_{canonical}(ep) \wedge \forall n \in$

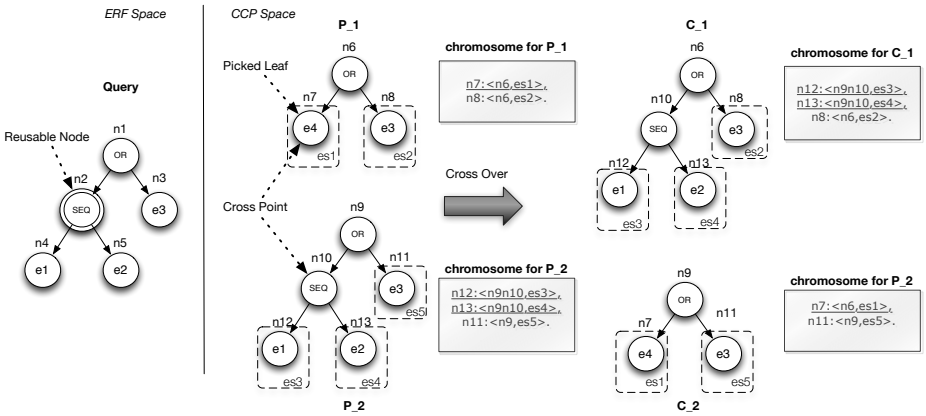


Fig. 2: Example of genetic encoding and crossover operation

$N_r, \exists ep^{'} \in ERF$, $ep^{'}$ is reusable to ep on n, as depicted in Figure 1. The ACPs for any ep can be enumerated by listing all possible combinations of the ACPs of their *immediate* reusable sub-patterns. By recursively aggregating those combinations, we derive the ACPs for ep.

4.2 Genetic Encodings for Event Syntax Trees

Given a CCP, we first assign global identifiers for the nodes in the event pattern ep of the CCP. Then we encode each leaf node in ep with its identifier, a service identifier referring to the service it represents and a string of identifiers indicating the path of the ancestor nodes of the leaf node, as shown in Figure 2.

4.3 Crossover and Mutation Operations

Given two genetically encoded parent CCPs P_1 and P_2, the event pattern specified in the query Q and the event reusability forest ERF, the crossover algorithm takes the following steps to produce the children (see example in Figure 2):

1. Pick randomly a leaf node l_1 from P_1, query the reusable relations stored in ERF to find the relevant reusable node n_r in Q.
2. Starting from l_1, search backwards along the prefix of l_1 and locate node $n_1 \in P_1$, such that the event pattern represented by $T(n_1) \subseteq P_1$[3] is a substitute to $T(n_r) \subseteq Q$, then mark node n_1 as the cross point for P_1.
3. For all leaf nodes in P_2, denoted L_2, find $l_2 \in L_2$ which are also reusable to Q on n_r, or on $n_r^{'}$ which is a descendant of n_r, then, mark the cross point $n_2 \in P_2$.

[3] given $n \in P$ is a node in pattern P, we denote $T(n) \subseteq P$ the sub-tree/sub-pattern of P with n as its root

Table 2: Brute-force enumeration vs. genetic algorithm

Test No.	CCP Size	Time (ms)	Avg. Utility	Max. Utility
BF-4	8004	5676	0.62	1.336
BF-5	27005	15429	0.624	1.771
BF-6	74074	37702	0.646	1.529
	Init. Population		**Init. Avg. Utility**	
GA-4	200	353	0.413	1.121
GA-5	200	649	0.442	1.324
GA-6	200	757	0.473	1.252
GA-50	200	1270	0.403	1.303

4. If $L_2 = \emptyset$, it means the sub event pattern $T(n_r) \in Q$ is not implemented *locally* in P_2, so there must be at least one leaf node $l_2 \in L_2$, such that the event pattern represented by $T(l_1) \subseteq P_1$ is reusable to the one represented by $T(l_2) \subseteq P_2$. For each such l_2, mark the relevant reusable node in Q as the new n_r, and try to find n_1 in the prefix of l_1 such that $T(n_1) \subseteq P_1$ is a substitute to $T(n_r) \subseteq Q$. If such n_1 is found, mark it as the new crossover point for P_1, similarly, mark the new cross point $n_2 \in P_2$.
5. If n_1 or n_2 is the root node, do nothing but keep the parents along with the new generations and give them a 100% chance of selection next time. Otherwise, swap the sub-trees in P_1, P_2 whose roots are n_1, n_2 (and therefore the relevant genes), resulting in two new CCPs.

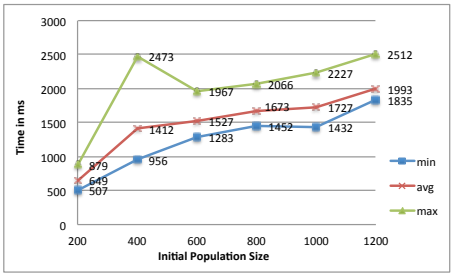
The mutation operation changes the composition plan for a leaf node in a CCP. To do that we select a random leaf node n in a CCP P, and treat the event pattern of n (possibly a primitive event) as an event query to be composed, then we use the same random CCP creation process specified in the population initialization (Section 4.1) to alter its implementation.

5 Evaluation

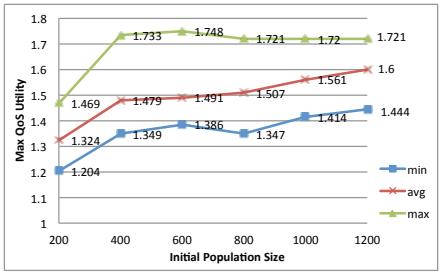
In this section we present the experimental results of the proposed approaches based on simulated datasets. The weights of QoS metrics in the preference vector are equally set to 1.0, and a loose constraint is defined in the query which do not reject any event service compositions to enlarge the search space.

5.1 Brute-Force Enumeration vs. Genetic Algorithm

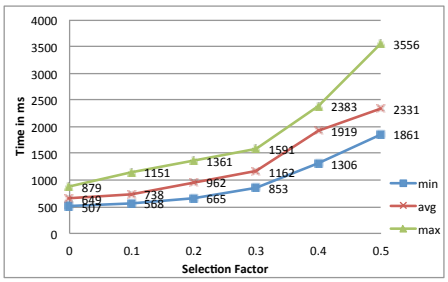
In the first experiment, we compare our genetic algorithm with brute-force enumerations in terms of the maximum QoS utility obtained and execution time required.We test both methods over three random event service repositories with



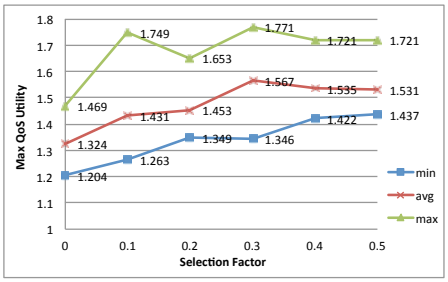
(a) Convergence time under various population size



(b) Max QoS utility under various population size



(c) Convergence time under different selection factor



(d) Max QoS utility under different selection factor

Fig. 3: Performances under different population sizes and selection factors

different sizes.The results are shown in Table 2 ("BF" and "GA" indicate the test for brute-force enumeration and genetic algorithm, receptively. The number after the dash is the number of candidate event services for each sub-event detection task). From the results in Table 2, we can see that GA based approach produces about 79% optimal results in much a shorter time, compared with the brute-force enumerations.

5.2 Convergence Time vs. Degree of Optimization

There are two ways to increase the utility in the GA results: increase the size of the initial population or the selection probability for the individuals in each generation. To evaluate the influence of the initial population size and selection probability, we execute the genetic evolutions with different population sizes and selection probabilities over the second dataset (BF-5) in Table 2.

Figure 3(a) and Figure 3(b) show the growth of execution time and best QoS utility retrieved using from 200 to 1200 CCPs as the initial populations. From the results we can see that the growth of evolution time is (almost) linear to the size of the initial population. In total, increasing the initial population from

200 to 1200 gains an additional 0.276 (15.6%) QoS utility with the cost of 1344 milliseconds of execution time.

In the tests above, we adopt the *Roullete Wheel* selection policy with *elites*. However, this selection policy results in early extinction of the population. To produce more generations, we simply increase the selection probability with a constant $0 \leq F \leq 1$, we call the additional F the *selection factor*. Figure 3(c) and Figure 3(d) show the execution time and best evolution results with different selection factors from 0 to 0.5.

6 Conclusions and Future Work

In this paper a QoS aggregation schema and utility function is proposed to calculate QoS vectors for event services (compositions) and rank them based on user-defined constraints and preferences. Then, a genetic algorithm is developed and evaluated to efficiently create optimal event service compositions. The experimental results show that the genetic algorithm is scalable, and by leveraging the trade-off between convergence time and degree of optimization, the algorithm gives 79% to 97% optimized results. As future work, we plan to validate our approach based on real-world datasets. We also plan to enable adaptive event compositions based on the GA developed in this paper.

References

1. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: A lightweight approach for qos-aware service composition. In: Proceedings of 2nd international conference on service oriented computing (ICSOC 04) (2004)
2. Gao, F., Curry, E., Bhiri, S.: Complex Event Service Provision and Composition based on Event Pattern Matchmaking. In: Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems. ACM, Mumbai, India (2014).
3. Hasan, S., Curry, E.: Approximate Semantic Matching of Events for The Internet of Things. ACM Transactions on Internet Technology (TOIT) (2014)
4. Hinze, A., Sachs, K., Buchmann, A.: Event-based applications and enabling technologies. In: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems. pp. 1:1–1:15. DEBS '09, ACM, New York, NY, USA (2009).
5. Jaeger, M., Rojec-Goldmann, G., Muhl, G.: Qos aggregation for web service composition using workflow patterns. In: Proceedings of the Eighth IEEE InternationalEnterprise Distributed Object Computing Conference. EDOC 04. pp. 149–159 (2004)
6. Karatas, F., Kesdogan, D.: An approach for compliance-aware service selection with genetic algorithms. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) Service-Oriented Computing, Lecture Notes in Computer Science, vol. 8274, pp. 465–473. Springer Berlin Heidelberg (2013)
7. Wu, Q., Zhu, Q., Jian, X.: Qos-aware multi-granularity service composition based on generalized component services. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) Service-Oriented Computing, Lecture Notes in Computer Science, vol. 8274, pp. 446–455. Springer Berlin Heidelberg (2013).
8. Zhang, L.J., Li, B.: Requirements driven dynamic services composition for web services and grid solutions. Journal of Grid Computing 2(2), 121–140 (2004).