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

Feng Gao[1], Edward Curry[1], Ali Intizar[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 event service candidates (or service compositions) best suit users' and applications' quality-of-service requirements. In this paper, we address this issue by first providing a quality-of-service aggregation schema for complex event service compositions and then developing a genetic algorithm to efficiently create optimal event service compositions.

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

1 Introduction

Recent developments in Sensor Networks and Information Communication Technologies along with cheap and fast Internet and reduced cost of sensors are seen as an enabler for the Internet-of-Things (IoT). Great opportunities are arising for rendering IoT-enabled services in "smart cities": smart city applications require more of such services to fulfill their promise in promoting urban performances in terms of sustainability, high quality of life and wiser management of natural resources [3]. For example in the city of Aarhus [3] traffic sensors, pollution monitors, parking meters, social feeds, library data and more are deployed in the city and the real time data are made publicly available as a rich source of knowledge to facilitate new services and applications, but we are not there yet.

Complex Event Processing (CEP) and event-based systems are important enabling technologies for smart cities [7], due to the need for integrating and processing high volumes of real time physical and social events. However, along

[3] Open Data Aarhus: `http://www.odaa.dk/`

with new opportunities, new challenges also arise. With the multitude of heterogeneous event sources to be discovered and integrated [6], it is crucial to determine how to best answer complex event requests and identify which event source should be considered to match specific quality requirements from users or applications [12].

Non-functional properties, e.g.: quality-of-service (QoS) properties, can play a pivotal role in guiding such selection if used as dimensions for finding the optimal event service composition plan that provides the best available results. Existing publish/subscribe based event systems and middleware use propriety event advertisement and subscription formats (which leads to silo architectures) and provide limited supports for non-functional requirements related to event subscriptions [10].

In this paper, we extend the work in [5], which aims to provide CEP applications as reusable services where reusability is determined by examining complex event patterns and primitive event types. The extension aims at enabling QoS-aware event service compositions in complex event-service networks. The conceptual architecture of such networks is illustrated in Figure 1.

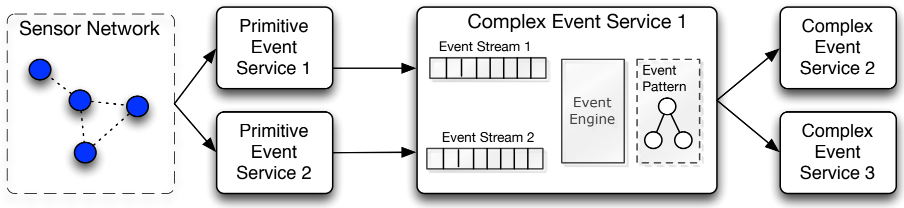


Fig. 1: Architecture of Complex Event Service Networks

In order to facilitate QoS-aware complex event service composition, two issues should be considered which we tackle in this paper: 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 for complex event service compositions for IoT because 1) there can be a large set of sensor devices or ICT services suitable for detecting an event and 2) a CEP service may reuse existing services at different granularity levels, resulting in different sets of event detection tasks involved.

In summary, this paper contributes to both aspects by:

- creating event QoS aggregation rules and utility functions used to estimate and assess QoS for event service compositions, and
- enabling efficient event service compositions and optimization w.r.t 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, e.g., in [1, 8, 13]. In this paper we extract some typical QoS properties from the existing work and define a similar utility function based on *Simple Additive Weighting* (SAW). However, the aggregation rules in the existing work focus on conventional web services rather than complex event services, which has a different QoS aggregation schema. For example, event engine also has an impact on the QoS aggregation, which is not considered in conventional service QoS aggregation. Also, the aggregation rules for some QoS properties based on event composition patterns is different to those based on workflow patterns (as in [8]), which we will explain in details in Section 3.2.

As a second step, different concrete service compositions are created are compared w.r.t. their QoS utilities to determine the optimal choice. To achieve this efficiently, various GA based approaches are developed. In [15] the chromosome encoded with binary bits representing whether a concrete service is selected or not. The problem with this approach is that the readability of the genomes are poor and the chromosome length is not fixed during evolution. In [2] the authors use a different encoding approach, which leads to a fixed chromosome length. In [14] a two-dimensional genome encoding is proposed to express all execution paths while considering task relations, but crossover and mutation needs validation. In [4], the authors use tree coding chromosome, crossovers operate on sub trees and mutation operate on leaf nodes to avoid invalid reproductions. In [9] the authors develop a GA based approach that goes beyond QoS-aware composition and enables compliance-aware service composition.

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 consists of different service tasks, i.e., service tasks on different granularity levels. A more recent work in [13] addresses this issue by presenting the concept of Generalized Component Services (GCS) and developing the GA encoding techniques and genetic operators based on GCS. Results in [13] 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, [13] only caters for *Input, Output, Precondition* and *Effect* (IOPE) based service compositions. Complex event service composition requires an *event pattern* based reuse mechanism [5]. As a result, it requires different genetic encoding mechanisms and crossover operations.

3 QoS Model and Aggregation Schema

The QoS properties of event service compositions may vary depending on the set of member event services used in the compositions. Some QoS properties may propagate along the event service network. In this section, a QoS model is used to represent some sample QoS properties. Then the QoS aggregation schema is presented to estimate the QoS properties for complex event service composition plans. Finally a utility function is introduced to evaluate the QoS performance under constraints and preferences.

3.1 Quality-of-Service Properties of Event Services

The event QoS attributes describe the non-functional performance of event services (and service compositions). In this paper, we do not intend to create a complete and precise QoS ontology for event services. Instead, some typical and important QoS attributes are investigated, including:

- *Latency* describes the delay in time for an event service, i.e., the temporal difference between the time when the event consumer receives the notification and the time when the event actually happens (usually denoted by the timestamp of the event), denoted L,
- *Price* describes the monetary cost for an event services, denoted P,
- *Energy Consumption* describes the energy cost for an event service, denoted E,
- *Bandwidth Consumption* describes the usage of network bandwidth for an event service, denote B,
- *Availability* describes the possibility of an event service being accessible, it can be numerically represented in percentages, denoted Ava,
- *Completeness* describes the completeness of events delivered by an event service, it can be numerically represented as recall rates in percentages, denoted C,
- *Accuracy* describes the correctness of events delivered by an event service, it can be numerically represented as precision in percentages, denoted Acc and
- *Security* describes the security protocol used by event services, it can be numerically represented as integer security levels, while bigger numerical value indicates higher security level, denoted S.

By the above definition, a quality vector $Q =< L, P, E, B, Ava, C, Acc, S >$ can be specified to indicate the performance of an event service in 8 dimensions.

3.2 QoS Aggregation

To calculate the overall QoS performance of an event service composition, a QoS aggregation schema is required. The QoS performance of an event service composition is considered to be influenced by three factors: *Service Infrastructure*, *Composition Pattern* and *Event Engine*. The *Service Infrastructure* consists of computational hardware, service I/O implementation and the physical network connection, it determines the inherent I/O capability of a service. The *Composition Pattern* refers to the local event patterns evaluated by the event engine and the set of member event services directly involved. Indeed, the performance varies on which services are used to produce the member events and how they are logically correlated by event operators. In this paper four event operators are considered: *And*, *Or*, *Sequence* and *Repetition.* The internal implementation of the *Event Engine* also has an impact on the event service composition performance. However, it can be difficult to assess or specify, because it depends on different implementations of event engines. Table 1 summarizes how different QoS parameters of an event service composition are calculated based on the three factors.

Table 1: Overall QoS calculation

Dimensions	**QoS Symbols**			**Overall Calculation**
	Service Infrastructure	Composition Pattern	Event Engine	
Latency	L_i	L_c	L_e	$L = L_i + L_c + L_e$
Price	P_i	P_c	n/a	$P = P_i + P_c$
Energy	E_i	E_c	E_e	$E = E_i + E_c + E_e$
Bandwidth	n/a	B_c	n/a	$B = B_c$
Availability	Ava_i	Ava_c	n/a	$Ava = Ava_i \times Ava_c$
Completeness	C_i	C_c	n/a	$C = C_i \times C_c$
Accuracy	Acc_i	Acc_c	Acc_e	$Acc = Acc_i \times Acc_c \times Acc_e$
Security	S_i	S_c	n/a	$S = \min(S_i, S_c)$

The *Composition Plan* is a key factor in aggregating QoS properties for event service compositions. As in [5], event patterns are represented as event syntax trees. In this paper, a step-wise transformation of event syntax tree is adopted to aggregate QoS properties. More specifically, we apply aggregation rules from the leaves to the roots on event syntax trees. Aggregation rules for different QoS dimensions can be event operator dependent or independent. Event operator dependent rules are defined based on the QoS properties of the set of Direct Sub-Trees (DSTs) of the entire event syntax trees. Event operator independent rules are defined based on the QoS properties of the set of Immediately Composed Event services (ICEs). Table 2 shows the detailed rules for each quality dimension. In the following we explain the rationale for each rule.

1. **Price** and **Energy Consumption** are operator independent properties. They can be specified in different manners, e.g., price can be charged over subscription time or volume, similar for energy consumption. For simplicity we assume they are specified over time. Then the overall price or energy cost

Table 2: QoS aggregation rules based on composition patterns

Dimensions	**Event Operators**			
	Repetition	Sequence	And	Or
$P_c(\mathscr{E}), E_c(\mathscr{E}) =$	$\sum P_c(e), \sum E_c(e)$, where $e \in \mathscr{E}_{ice}$			
$Ava_c(\mathscr{E}) =$	$\prod Ava_c(e)$, where $e \in \mathscr{E}_{ice}$			
$S_c(\mathscr{E}) =$	$min\{S_c(e), e \in \mathscr{E}_{ice}\}$			
$L_c(\mathscr{E}) =$	$L_c(e), e$ is the last event in $\mathscr{E}_{dst}$		$avg\{L_c(e), e \in \mathscr{E}_{dst}\}$	
$C_c(\mathscr{E}) =$	$\frac{min\{C_c(e) \cdot f(e), e \in \mathscr{E}_{dst}\}}{card(\mathscr{E}) \cdot f(\mathscr{E})}$			$\frac{max\{C_c(e) \cdot f(e), e \in \mathscr{E}_{dst}\}}{f(\mathscr{E})}$
$Acc_c(\mathscr{E}) =$	$\frac{card(\mathscr{E}) \cdot f(\mathscr{E})}{min\{Acc_c(e)^{-1} \cdot f(e), e \in \mathscr{E}_{dst}\}}$			$\frac{f(\mathscr{E})}{max\{Acc_c(e)^{-1} \cdot f(e), e \in \mathscr{E}_{dst}\}}$

of a certain event $\mathscr{E}$ is the sum of the price or energy cost of the *immediately composed event* services (denoted $\mathscr{E}_{ice}$).

2. **Availability** and **Security** are operator independent properties. The availability of $\mathscr{E}$ is the product of event service availability in $\mathscr{E}_{ice}$; the security level is determined by the most vulnerable event service in $\mathscr{E}_{ice}$.
3. **Latency** of event $\mathscr{E}$ is an operator dependent property. It is determined by the last event completing the event pattern of $\mathscr{E}$. Therefore, if the root operator of $\mathscr{E}$ is sequence or repetition, the latency of $\mathscr{E}$ is same as the last event in the *direct sub events* of $\mathscr{E}$ (denoted $\mathscr{E}_{dst}$). Otherwise, it is hard to predict when the last direct sub event occurs, therefore we make an approximation with the average of the latencies of the event services in $\mathscr{E}_{dst}$.
4. **Completeness** and **Accuracy** are operator dependent properties. The reliability of $\mathscr{E}$ can be estimated based on its *direct sub event frequencies* (denoted $f(e), e \in \mathscr{E}_{dst}$, for the estimation of $f(e)$ see [5][4]) and direct sub event reliabilities. The idea is to derive the estimated receiving frequency of $\mathscr{E}$ by investigating its direct sub event sending frequencies and reliability, and then divide the estimated receiving frequency by the estimated logical frequency (the theoretical value of how often should $\mathscr{E}$ occur). For *Sequence*,*Repetiton*[5] or *And* operators, the estimated receiving frequency of $\mathscr{E}$ is the minimum of the products of the sending frequency and reliability of the event services in $\mathscr{E}_{dst}$. On the contrary, for *Or* operators the maximum is used as the estimated receiving frequency. Based on the similar ideas on frequency estimation, aggregation rules for accuracy can be derived.
5. **Bandwidth Consumption** can be measured by the number of events consumed by an event composition, i.e., its traffic demand. We refer readers to [5] for detailed description on estimating traffic demands of event composition based on the frequencies of primitive events.

[4] It is not entirely accurate to reuse the frequency estimation method directly in this context, because it was defined without considering the impact of reliability and accuracy, however for brevity we omit the modifications required.

[5] The function $card(\mathscr{E})$ gives the cardinality of the repetition, other operators have a default cardinality of 1.

3.3 Event QoS Utility Function

To choose the best service composition under users' constraints and preferences, a QoS utility function is needed. In this paper, a SAW based approach with normalization is used to calculate the QoS utility. Given a quality vector of an event service composition $Q =< L, P, E, B, Ava, C, Acc, S >$ representing the service QoS capability, we denote $q \in Q$ as a quality value in the vector, $O(q)$ as the theoretical optimum value in the quality dimension of q, $C(q)$ as the user-defined value specifying the hard constraints on the dimension and $0 \leq W(q) \leq 1$ as the weighting function of the quality metric, representing users' preferences. Further, we 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). Then, the QoS utility U is derived by:

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

where $q_i \in Q_+, q_j \in Q_-$. According to Equation (1) the best event service composition should have the maximum utility U.

4 Genetic Algorithm for QoS-Aware Event Service Composition Optimization

The detection of the complex event pattern of an event service composition can be achieved by monitoring different sets of member events on different granularity levels. Each member event detection task can be achieved by subscribing to a set of event services. If a complex event pattern can be detected by n different sets of sub-events, each set has an average size of m sub-events, and each sub-event detection task can be implemented by subscribing to l (on average) event service candidates, the total number of concrete composition plans is estimated to be $n \cdot l^m$. Clearly, in large scale scenarios, it is highly inefficient to enumerate all possible compositions of event services and evaluate their overall performances. In this paper, we propose a heuristic method based on *Genetic Algorithms* (GA) to derive global optimizations for event service compositions, without the need for enumerating all possible composition plans.

Typically, a GA requires a genetic encoding for the solution space, as well as a fitness function to evaluate the solutions. A standard GA based search iterates the procedure of *select*, *crossover* and *mutate* until termination conditions are met. The GA approach in this paper also follows these steps. 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 in [5], where hierarchies of reusable event services are maintained, called an Event Reusability Forest (ERF).

An ERF consists of a set of Event Reusability Hierarchies (ERH). An ERH is a *directed-acyclic-graph*, denoted $ERH = (T, R)$ where T is a set of canonical event patterns[6] of event services, $R \subset T \times T$ is a set of edges (reusable relations) connecting nodes. Generally speaking, an ERF serves as a reusability index for (complex) event services based on their event patterns. In the following we elaborate how we utilize the ERF in the genetic algorithms.

4.1 Population Initialization

Given an event service composition request represented by 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 event 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 *Concrete Composition Plans* (CCPs) with random structure and service bindings are obtained. The second and third steps are trivial, next we explain how ACPs are derived based on ERF.

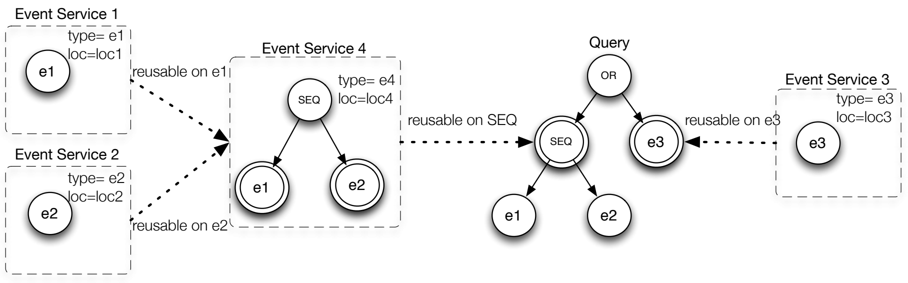


Fig. 2: Marking the reusable nodes

When an event pattern ep is inserted into the ERF, we can mark its *reusable nodes* denoted N_r: $N_r \subseteq f_{canonical}(ep) \wedge \forall n \in N_r, \exists ep^{'} \in ERF$, $ep^{'}$ is reusable to ep on n, as depicted in Figure 2. Obviously, a primitive event involved in ep has at most 1 ACP, which is subscribing to the primitive event services with the requested primitive event type. And the ACPs for any sub-event patterns of ep (including ep itself) can be enumerated by listing all possible combinations of the ACPs of their *immediate* reusable nodes. By recursively aggregating those combinations, we can derive the ACPs for ep.

[6] In [5] the function $f_{canonical}(ep)$ is defined to derive the canonical form for an event pattern ep.

It is worth noticing that although it requires enumerating *all* ACPs to ensure the diversity in the structure of event compositions, the size of the different combinations of reusable sub-events is moderate, compared to the size of all concrete composition plans. Meanwhile, the reusable relations can be efficiently retrieved from the ERF [5]. Therefore, the enumeration of ACPs can be done efficiently.

4.2 Genetic Encodings for Event Syntax Trees

With the ability to initialize the population, now we need to genetically encode the individuals in the population to represent their various characteristics. In a typical encoding for service compositions, each service task is encoded with a value indicating the concrete service implementing the task. These values are ordered in a sequence so that the positions of the values indicate which service tasks they relate to. Similarly, we encode the event detection tasks (leaf nodes) in an CCP with values to indicate the service bindings used. However, the positions of the values (arranged in any tree traversal orders) cannot represent which parts of the event detection task do the reused event services contribute in, since the CCPs are unordered trees with variable structures. The only thing identifying an event detection task is the event pattern it detects.

Nevertheless, the sequence of ancestors of the nodes can give a hint about which roles they play in the entire event pattern and reducing the search space while finding their functional equivalent counter-parts. Therefore, we first assign a global identifier for all the nodes in the CCPs generated during population initialization, as well as the nodes in the event patterns in the ERF. Then we encode the leaf nodes (reusable nodes) in CCPs with a string of node identifiers as a prefix representing the path of its ancestor nodes and a service identifier indicate the concrete service binding, as shown in Figure 3.

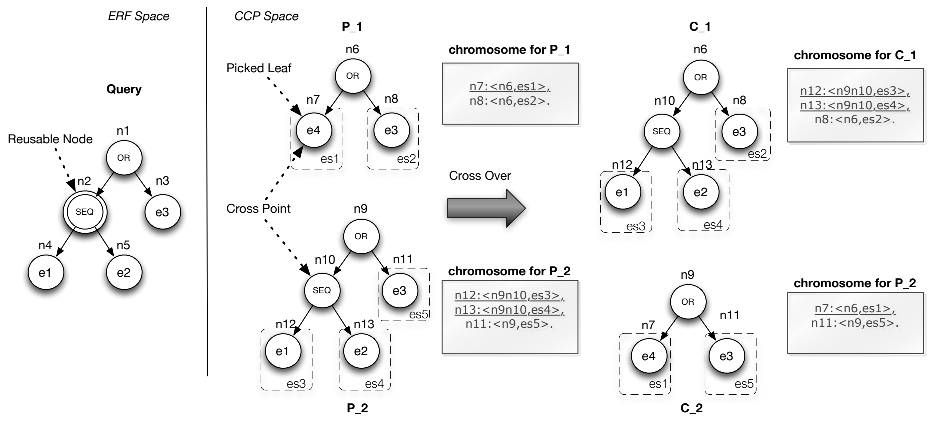


Fig. 3: Example of genetic encoding and crossover operation

4.3 Crossover and Mutation Operations

After the population initialization and encoding, the preparation tasks for GA based optimization are completed. The algorithm iterates the cycle of select, crossover and mutate to find optimal solutions. The selection is trivial, individuals with better finesses (QoS utility derived by Equation (1)) are more likely to be chosen to reproduce. In the following we explain the details on crossover and mutation operations for event service composition optimization.

Crossover To ensure valid child generations are produced by the crossover operation, parents must only exchange genes representing the same part of their functionalities, i.e., the same (sub) event detection task, specified by semantically equivalent event patterns. An example of crossover is illustrated in Figure 3. 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:

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 event pattern represented by $T(n_1) \subseteq P_1$ 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 $n_2 \in P_2$.
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.

Mutation The mutation operation changes the composition plan for a leaf node in a CCP. To do that we can simply select a random leaf node n in a CCP P, and treat the event pattern of n (possibly a primitive event) as a new event query that needs 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. We first present the experiment setups and the datasets used in the experiments, then, we compare the results and execution time of a brute-force enumeration and the proposed genetic evolution, finally, we show the means of improving genetic evolution results at the cost of slower convergence rate.

5.1 Experiment Settings and Datasets

All experiments are carried out on a Macbook Pro with a 2.53 GHz duo core cpu and 4 GB 1067 MHz memory. Prototypes are developed in Java. The Java Virtual Machine is configured with a minimal heap size of 64 MB and a maximal heap size of 256 MB. All results are derived from 10 test iterations.

In the experiments the event pattern of the query to be composed consists of 10 nodes, including 6 primitive events and 4 event operators. This event pattern is also used to create event service descriptions stored in the event service repository: for each sub-tree of the pattern (including the entire tree and leaf nodes), a number of (complex) event service descriptions are created with random QoS vectors. These event service descriptions are reused to create event service compositions. The QoS utility is calculated for each event service composition according to the QoS utility function defined in Equation (1). The weights of QoS metrics representing the QoS preferences of the query are equally set to 1.0, and a loose constraint is defined in the query which do not reject any event service compositions in order to enlarge the search space. The performance of the GA based composition and a brute-force enumeration is recorded and compared.

5.2 Brute-Force Enumeration vs. Genetic Algorithm

In the first experiment, we compare our genetic algorithm with the brute-force enumerations in terms of maximum QoS utility obtained and execution time required. First, we build three event service repositories with different sizes. Creating 4 event services for each sub-pattern of the query leads to 8004 CCPs, 5 event services per pattern gives 27005 CCPs, 6 event services per pattern gives 74074 CCPs, 7 event services (and above) cannot be enumerated due to the memory limit. We observe the time used during the execution and capture the CCP with the highest QoS utility, then, we execute the genetic algorithm over these three datasets and compare the results with brute-force enumerations. The results are shown in Table 3. In the test names, "BF" indicates the test uses brute-force enumeration, "GA" indicates the genetic algorithm is used for the test, the number after the dash represent the number of event services created for each sub-pattern. Throughout our evaluation, the crossover rate of the genetic algorithm is set to 100% and mutation rate is 5%.

From the results in Table 3, we can see that the execution time of brute-force enumeration grows exponentially to the number of service candidates for

Table 3: Brute-force enumeration vs. genetic algorithm

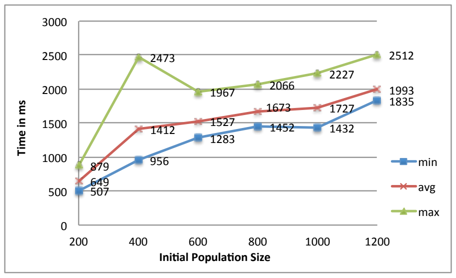
Test No.	CCP Size	Time (ms)	Avg. Utility	Max. Utility
BF-4	8004	5676	0.62	1.336
BF-5	27005	15429	0.624	1.771
BF-6	74074	37702	0.646	1.529
	Init. Population		**Init. Avg. Utility**	
GA-4	200	353	0.413	1.121
GA-5	200	649	0.442	1.324
GA-6	200	757	0.473	1.252
GA-50	200	1270	0.403	1.303

each sub-event, which is not scalable. However, the GA based approach can produce about 79 percent optimal results (compared to the global optimizations enumerated) within 353 to 757 milliseconds, using 200 random CCPs as the initial population. We run an additional genetic evolution over a 50 candidate per pattern event repository (which is far beyond the capacity of brute-force enumeration), and the evolution completes in 1270 milliseconds. The average utility of the initial population is lower than the entire set of CCPs is due to the fact that each ACP has an equal chance of producing CCPs during the initialization, but some ACPs create more CCPs with higher QoS utility in the enumeration.

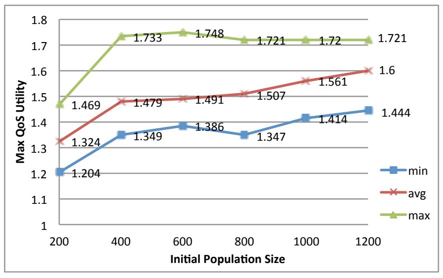
5.3 Convergence Time vs. Degree of Optimization

If a user is not satisfied with the above results (79% optimal), he may sacrifice some execution time in exchange for a higher utility. There are two ways to do that: increase the size of the initial population or the selection probability of 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 3. Figure 4(a) and Figure 4(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, and by increasing the population size we always get better results. It is also observed that increasing the population size up to 400 has the fastest utility growth rate. The influence of increasing population size in GA is analyzed in [11]. In total, increasing the initial population from 200 to 1200 gains 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: for each individual i, the selection probability $P_i = \frac{U_i}{U_{max}}$, where U_i is the QoS utility of

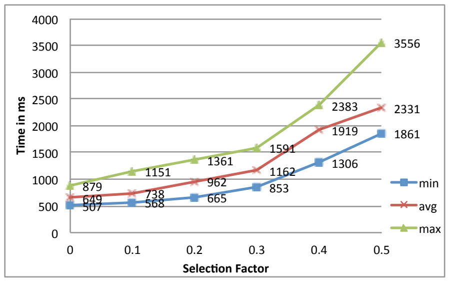


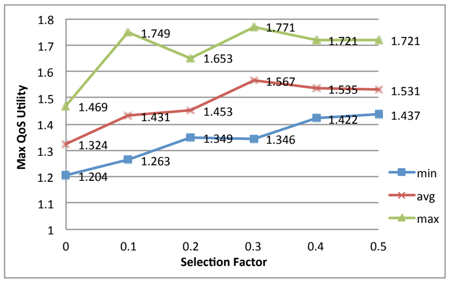
(a) Convergence time



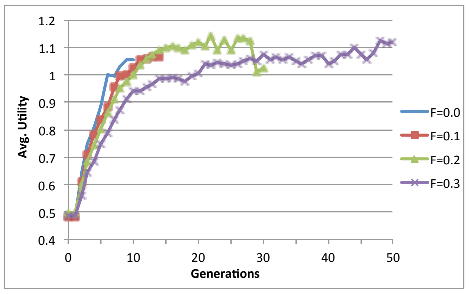
(b) Max QoS utility

Fig. 4: Performances under different initial population size





(a) Convergence time (b) Max QoS utility



(c) Average QoS utility

Fig. 5: Performances under different selection factor

i and U_{max} is the maximum utility in the generation. 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 5(a) and Figure 5(b) show the execution time and best evolution results with different selection factors from 0 to 0.5. Figure 5(c) show the average utility of each generation using different selection factors.

The results in Figure 5(a) show that the execution time increases when more individuals are selected in each generation, and the rate get faster as the selection factor gets bigger. The results in Figure 5(b) show that increasing the selection factor does not always bring better results: the best evolution result is achieved when $F = 0.3$, accepting more (bad) individuals after that may give worse evolution results. When $F = 0.3$, the genetic algorithm gains additional 0.243 (13.7%) QoS utility with the cost of 513 milliseconds of execution time. It is evident that increasing the selection probability (up to $F = 0.3$) is more efficient than increasing the size of the initial population. The user may also use a hybrid approach, using 1200 individuals in the initial population with the selection factor $F = 0.3$, the genetic algorithm finds a CCP with 1.721 (97.2% optimal) QoS utility within 2526 milliseconds.

6 Conclusions and Future Work

In this paper a QoS aggregation schema is proposed to calculate the overall QoS vector for an event service composition. Based on a user-defined constraint and weight vector, a QoS utility function is defined to calculate the degree of optimization for event composition. Finally, a genetic algorithm is developed and evaluated to efficiently create optimal event service compositions. A tree encoding schema for event service compositions and a crossover operation based on the Event Reusability Forest is presented. 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 carry out experiments to validate our QoS aggregation schema based on real-world datasets. We also plan to observe how the proposed genetic algorithm performs on real data. Moreover, we will investigate how other parameters, e.g.: crossover rate, mutation rate and other selection policies, may influence the performance of the algorithm.

Acknowledgments

This work is supported by the EU FP7 CityPulse Project under grant No.603095. http://www.ict-citypulse.eu

References

1. Alrifai, M., Risse, T.: Combining global optimization with local selection for efficient qos-aware service composition. In: Proceedings of the 18th international conference on World wide web. pp. 881–890. WWW '09, ACM (2009), `http://doi.acm.org/10.1145/1526709.1526828`
2. 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)

3. Caragliu, A., Del Bo, C., Nijkamp, P., et al.: Smart cities in Europe. Vrije Universiteit, Faculty of Economics and Business Administration (2009)
4. Gao, C., Cai, M., Chen, H.: Qos-aware service composition based on tree-coded genetic algorithm. In: Proceedings of the 31st Annual International Computer Software and Applications Conference - Volume 01. pp. 361–367. COMPSAC '07, IEEE Computer Society, Washington, DC, USA (2007), http://dx.doi.org/10.1109/COMPSAC.2007.174
5. 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), http://dx.doi.org/10.1145/2611286.2611287
6. Hasan, S., O'Riain, S., Curry, E.: Approximate semantic matching of heterogeneous events. In: Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems. pp. 252–263. DEBS '12, ACM, New York, NY, USA (2012), http://doi.acm.org/10.1145/2335484.2335512
7. 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), http://doi.acm.org/10.1145/1619258.1619260
8. 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)
9. 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), http://dx.doi.org/10.1007/978-3-642-45005-1_35
10. Mahambre, S.P., S.D., M.K., Bellur, U.: A taxonomy of qos-aware, adaptive event-dissemination middleware. IEEE Internet Computing 11(4), 35–44 (2007)
11. Roeva, O., Fidanova, S., Paprzycki, M.: Influence of the population size on the genetic algorithm performance in case of cultivation process modelling. In: Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on. pp. 371–376 (Sept 2013)
12. Verginadis, Y., Patiniotakis, I., Papageorgiou, N., Stuehmer, R.: Service adaptation recommender in the event marketplace: conceptual view. In: The Semantic Web: ESWC 2011 Workshops. pp. 194–201. Springer (2012)
13. 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), http://dx.doi.org/10.1007/978-3-642-45005-1_33
14. Zhang, C., Su, S., Chen, J.: A novel genetic algorithm for qos-aware web services selection. In: Proceedings of the Second international conference on Data Engineering Issues in E-Commerce and Services. pp. 224–235. DEECS'06, Springer-Verlag, Berlin, Heidelberg (2006), http://dx.doi.org/10.1007/11780397_18
15. 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), http://dx.doi.org/10.1007/s10723-004-4202-1