International Journal on Semantic Web and Information Systems
Volume 12 • Issue 4 • October-December 2016

QoS-Aware Stream Federation and Optimization Based on Service Composition

Feng Gao, Insight Centre for Data Analytics, National University of Ireland, Galway, Ireland
Muhammad Intizar Ali, Insight Centre for Data Analytics, National University of Ireland, Galway, Ireland
Edward Curry, Insight Centre for Data Analytics, National University of Ireland, Galway, Ireland
Alessandra Mileo, Insight Centre for Data Analytics, National University of Ireland, Galway, Ireland

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 in urban 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. A sub-optimal service composition may lead to inaccurate event detection, lack of system robustness etc. In this paper, the authors address these issues by first providing a quality-of-service aggregation schema for complex event service compositions and then developing a genetic algorithm to efficiently create near-optimal event service compositions. The authors evaluate their approach with both real sensor data collected via Internet-of-Things services as well as synthesised datasets.

KEYWORDS

Complex Event Service, Genetic Algorithm, Internet-of-Things, Quality-of-Service, Semantic Web, Service Composition, Service Optimisation, Service-Oriented Computing, Smart City

1. INTRODUCTION

Recent developments in the Internet-of-Things (IoT) envision "Smart Cities" and promise to improve urban performances in terms of sustainability, high quality of life and wise management of natural resources. Complex Event Processing (CEP) and event-based systems are important enabling technologies for smart cities (Hinze, Sachs, & Buchmann, 2009), 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 (Hasan & Curry, 2014), it is crucial to determine which event services should be used and how to compose them to match non-functional requirements defined by users or applications. Indeed, non-functional properties, e.g.: Quality-of-Service (QoS) properties, can play a pivotal role in service composition (Wu, Zhu, & Jian, 2013).

In (Gao, Curry, & Bhiri, 2014), CEP applications are provided as reusable services and the reusability of those event services is determined by examining event patterns. In this paper, we aim

DOI: 10.4018/IJSWIS.2016100103

to enable a QoS-aware event service composition and optimization. In order to facilitate this, two issues should be considered: QoS aggregation and composition efficiency. The QoS aggregation for a complex event service relies on how its member event services are correlated and composed. 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.

This paper is an extension of the work in (Gao, Curry, Ali, Bhiri, & Mileo, 2014). The extension is three-fold: 1) we introduce a realistic scenario in the context of traffic management using smart city applications with both real-world and synthetic datasets, 2) we provide in-depth analysis of the parameters used in the genetic algorithm using the datasets and 3) we validate the QoS aggregation rules with simulation. The remainder of the paper is organized as follows: Section 2 presents the a real-world travel planning scenario for the City of Aarhus as motivation, Section 3 introduces some preliminary concepts adopted in this paper; Section 4 presents the QoS model we use and the QoS aggregation rules we define; Section 5 presents the heuristic that enables QoS-aware event service compositions based on Genetic Algorithms (GA); Section 6 evaluates the proposed approach; Section 7 discusses related works in QoS-aware service planning before Section 8 concludes.

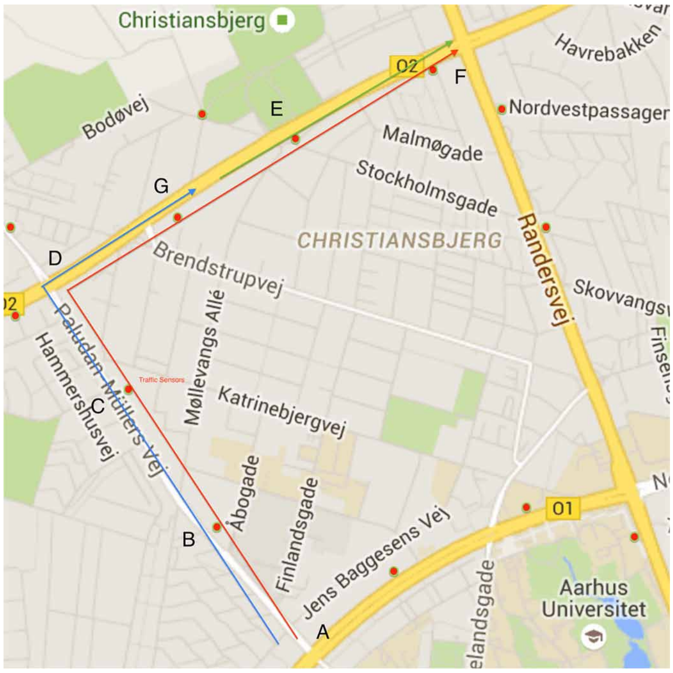
2. MOTIVATION SCENARIO

Event services can be applied in different Smart City scenarios. In this paper, we study the scenario of smart urban travel planning, as regular citizens, developers and city stakeholders rank it highly[1]. The city of Aarhus in Denmark has deployed a set of street-level traffic sensors. These sensors are paired as start nodes and end nodes. Each pair is capable of monitoring the average vehicle speed v and vehicle count n on a street segment (from the start node to the end node). Combined with the distance d between the two sensors, the estimated travel time $t = d/v$ and congestion level $c = n/d$ can be derived and published regularly as traffic report events.

Figure 1 shows some traffic sensor nodes (depicted as red dots) on the Aarhus city map. Suppose a user, Alice, has an important appointment in 15 minutes, and she has to travel from home (on segment A in Figure 1) to her work place (on segment F in Figure 1) within the time frame. Alice decides not to pick a route randomly since it is rush hour and there's a good chance that she may experience traffic congestion. Instead, Alice uses a travel planner application on her smartphone to select the fastest route. Alice would like to receive live traffic condition reports during her trip in case some traffic incidents happen on the selected route and a detour is necessary. To do that, she specifies the start and end location of the travel. She also wants to be sure that the time estimation is accurate, so she sets some non-functional constraints such as the accuracy of the estimated travel time above 90%.

According to Alice's request, the backend system will calculate possible routes and query the sensor services for the latest traffic condition. Based on this information, the system finds the fastest route (A-D-F) for Alice. Taking into account the QoS constraints specified by Alice, i.e., the accuracy of the estimated travel time to be above 90%, the system will try to create event service compositions reusing different available event services. For example, as shown in the map in Figure 1, if other users, e.g., Bob and Charlie, are living in the neighbourhood and they have deployed some semi-permanent services monitoring the traffic conditions within the same segments (B-C-G and E-F) on the route that Alice chooses and they have registered these services to the service registry, the backend system will recognise these services as reusable and try different combinations to find the optimal solution. Meanwhile, the optimization needs to be efficient to enable real-time re-planning and adapt to the fast changing service environment. Simply enumerating all possible composition plans will not scale.

Figure 1. Traffic sensors in Aarhus City



3. FOUNDATIONS

In this section, we briefly introduce the core concepts and methodologies for enabling pattern based event service composition (F. Gao, Curry, & Bhiri, 2014). These concepts are reused in this paper for enabling a constraint aware event service composition based QoS.

3.1. Conceptualization

The conceptualization of an event service in this paper is built upon existing concepts in literature. Table 1 shows the definitions of the terms used.

Table 1. Definitions of terms

Concepts	Definitions
Event	"An occurrence within a particular system or domain..." – (Etzion & Niblett, 2010).
Complex Event	"An event consisting several different event instances" – (Etzion & Niblett, 2010) "An event that summarises, represents, or denotes a set of other events." – *EPTS*[2]
Event Pattern	"A template containing event templates, relational operators and variables." –*EPTS*
Service	"A service is a self-contained, logical representation of a repeatable business activity that has a specified outcome," "is a 'black box' to the consumer of the service" – *The Open Group*[3]
Event Service	An asynchronous notification service that accepts subscriptions from event consumers and delivers events.
Complex Event Service (CES)	An event service that delivers complex events with the event patterns published as part of its service description.
Primitive Event Service (PES)	An event service not equipped with CEP capability or does not describe the event pattern in the service description, i.e., an event service that is not a CES.

The definition of "events" is usually broad. In this paper, we adopt the definition in (Etzion & Niblett, 2010) and consider any occurrence/arrival of data/information in the system an event instance. An example of an event could be a measurement reported by the traffic sensor as described in Section 2. A complex event consists of other event instances and an event pattern describes the rules to be evaluated against those event instances in order to detect the complex event. An example of a complex event could be a traffic jam detected using multiple traffic sensor measurements, and the event pattern could be all traffic sensors in a region are reporting high vehicle count and low vehicle speed repeatedly over the last 10 minutes.

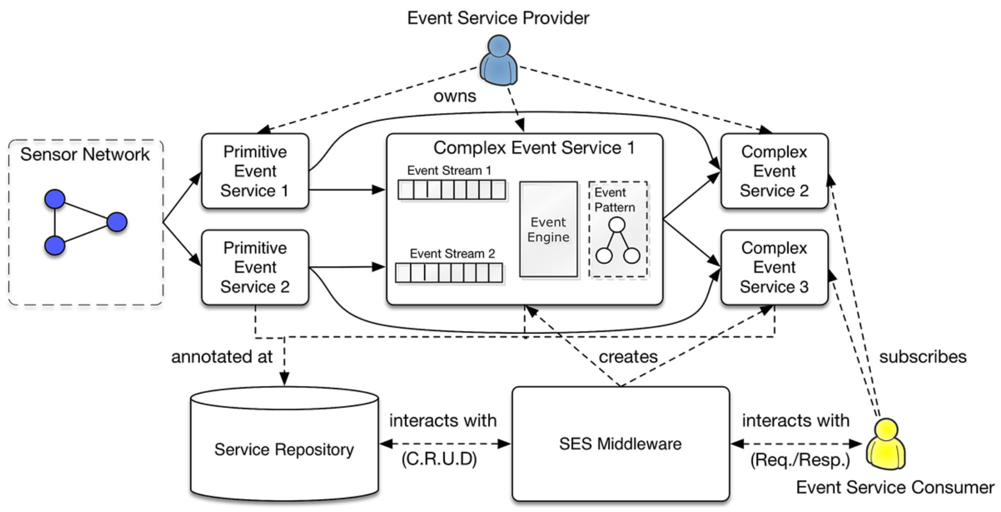
When the sources generating event messages are wrapped as services following the Service-Oriented Computing paradigm, we call them event services. Like Web Services, event services describe their service capability in a service repository to facilitate service discovery, composition and invocation. Suppose an event/stream processing system detecting the aforementioned traffic jams is implemented as an event service, we should describe its capability with the event pattern, since it gives the exact semantics of this complex event and is essential for identifying the service capability. We call this service a Complex Event Service (CES). When an event service does not describe its complex event pattern in the service description, it is a Primitive Event Service (PES), an example of PES would be the traffic sensor service publishing traffic reports (with attribute-value pairs) when no specifications are provided on how these reports are generated.

Figure 2 shows the overview of an event service network and illustrates how CESs are composed with PESs, which collect sensor readings from a sensor network and deliver sensor observation events. It also illustrates how event service consumers and providers interact with service repository and middleware to discover and compose event services.

3.2. Event Service Modelling and Matchmaking

Event service descriptions are crucial for enabling event service discovery and composition. In this paper, we use the Complex Event Service Ontology (CESO)[4] to semantically describe CESs. CESO is an extension of OWL-S[5]. In CESO, an event service is described with an Event Grounding and Event Profile. The concept of Event Grounding is similar to Service Grounding in OWL-S. It tells an event consumer how to access the event service by providing information on service endpoints and message formats. An Event Profile is comparable to the Service Profile in OWL-S, which describes the functional and non-functional aspects of an event service.

Figure 2. Overview of an event service network



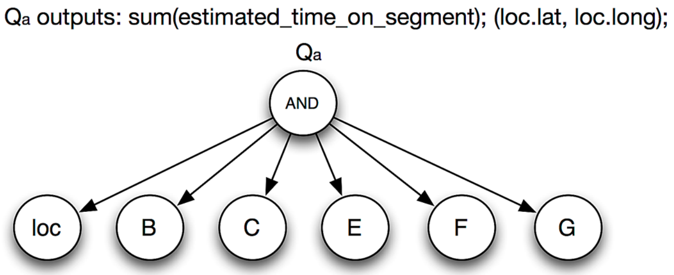
Event profiles are key documents used for event service discovery and composition. An event profile is a tuple $\mathcal{E} = (t, ep, D, Q)$, where *t* is the domain-specific event type, *ep* is the event pattern describing the temporal and logical rules for detecting complex events, *D* is the data payloads of the event and *Q* is the QoS parameters of the event service. An event service request is a tuple $\mathcal{E}_r = (t_r, ep_r, D_r, Const, Pref)$, where t_r is the requested event type, ep_r is the requested event pattern, D_r is the requested data payloads, *Const* is a set of QoS constraints and *Pref* is a set of QoS preferences. Like other Web Service ontologies, CESO provides a common ground for defining the concepts used in event services and allows for querying and reasoning over these concepts. For example, the SPARQL query in Listing 1 can be used to discover PESs with the requested event type (temperature measurement) and payloads (temperature sensor reading). With RDFS level reasoning, it can also find PESs annotated with terms subsumed by the requested terms, e.g., a PES backed by a specific type of temperature sensor.

Event patterns describe the temporal and logical correlations (using event operators, e.g., And, Or, Sequence and Repetition) between events (using event declarations). They can be represented as partially ordered trees, called Event Syntax Trees (ESTs). An example of EST describing Alice's travel planning request (See Section 2) is depicted in Figure 3[6]. This request describes a conjunction pattern between Alice's current location update event and several traffic report events: whenever all events are captured, calculate the sum of the estimated travel time for Alice using the traffic event payloads, and report her current location. Alice's non-functional constraints (e.g., accuracy of estimated time above 90%) and preferences can be annotated with CESO. Using the results calculated upon this request pattern, a post-processing method (could be implemented by a separate program) can determine if the estimated travel time is above some predefined threshold and if so, notify Alice, or suggest a new route. Also, when Alice moved beyond some segments, remove that segment from the query pattern and deploy the new query.

Listing 1. Discovering PES via SPARQL query

```
PREFIX ces: <http://www.insight-centre.org/ces#>
PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn>
PREFIX owls: <http://www.daml.org/services/owl-s/1.2/Service.owl#>
SELECT ?eventService
 WHERE { ?eventService owls:presents ?profile.
          ?profile owls:hasServiceCategory:TemperatureMeasurementService.
          ?eventService ces:hasPhysicalSource ?sensor. ?sensor ssn:observes ?property.
          ?property a:AirTemperature. ?property ssn:isPropertyOf:region_01.
```

Figure 3. Traffic planning event request for Alice (denoted Q_a)


Qa outputs: sum(estimated_time_on_segment); (loc.lat, loc.long);
Qa
AND
loc
B
C
E
F
G

The event patterns in CESO are annotated as nested RDF containers. CESO provides information on the provenance of event patterns using a transitive property *hasSubPattern* over patterns. Rule 1 in Listing 2 is used in CESO to entail sub-pattern relationships. Notice that *rdfs:member* is the super-property for the container membership property (i.e., rdf:_1, rdf:_2 ...) in RDF Schema version 1.1. The query in Listing 3 shows how to track the provenance within a single Event Profile. To track provenance relations between different event services, the additional Rule 2 in Listing 2 is required.

Using SPARQL over CESO provides inference capability over taxonomical and provenance information between event patterns and thus improves data interoperability. However, it does not fully address pattern-based pattern composition and QoS-aware optimization for CESs. Semantic equivalence of event patterns (denoted $ep_1 \doteq ep_2$) needs to be examined by comparing the tree isomorphism between canonical forms of event patterns. A canonical event pattern is derived by first expanding the leaves of an EST until all leaves are primitive events, and then removing all redundant event operators. We use $f_{canonical}$ to denote the function that derives canonical event patterns.

3.3. Pattern-Based Event Service Reusability Index

Comparing canonical ESTs can determine if two event pattern match. When no matches for an event request can be found, we can divide the requested EST into sub-trees and try to create an event service composition using matches of the sub-trees. An example of an event service composition (for a slightly advanced version of Bob's event request in Section 2) based on event pattern matching is illustrated in Figure 4. In this figure, Bob's request pattern is depicted as the EST on the top-left, and existing event services (1 to 4) are depicted on the right. Using these as input, the composition algorithm creates a composition plan on the bottom-left for Bob's request, which consists of a traffic congestion monitoring service (Event Service 4) and a road blockage notification service (Event Service 1).

However, creating event service compositions through identifying matches for sub-trees can be expensive and it will not scale because of the combinatorial explosion. To accelerate event service compositions, a reusability index is proposed in (F. Gao, Curry, & Bhiri, 2014) based on a binary *reusable* relation over event patterns. For two canonical event patterns ep_1 and ep_2, ep_1 is *reusable* (denoted $R(ep_1, ep_2)$) to ep_2, if and only if it is *directly reusable* (denoted $R_d(ep_1, ep_2)$) or *indirectly reusable* (denoted $R_i(ep_1, ep_2)$) to the other. Informally, $R_d(ep_1, ep_2)$ holds if ep_1 matches a sub-tree of the EST of ep_2, and $R_i(ep_1, ep_2)$ holds if ep_1 can partially replace a sub-tree of the EST of ep_2. Figure 5 shows how the pattern in Q_b can reuse other patterns.

Listing 2. Rules to entail sub-patterns

```
Rule1: (?x rdfs:member ?y) -> (?x ces:hasSubPattern ?y)]
[Rule2: (?ep1 ces:hasSubPattern ?s) (?s owls:presents ?p) (?p ces:hasPattern ?ep2)
        -> (?ep1 ces:hasSubPattern ?ep2)]
```

Listing 3. Tracking pattern provenance via SPARQL

```
SELECT ?subpattern
 WHERE {
        :SampleService owls:presents ?sampleProfile.
        ?sampleProfile ces:hasPattern ?pattern.
        ?pattern ces:hasSubPattern ?subPattern. }
```

Figure 4. Example of a composition plan for the advanced Bob's request (denoted by Q_b): Notify the user when there are traffic congestions (labelled "cng") on route B to G, or if a road construction has blocked route (labelled "blk")

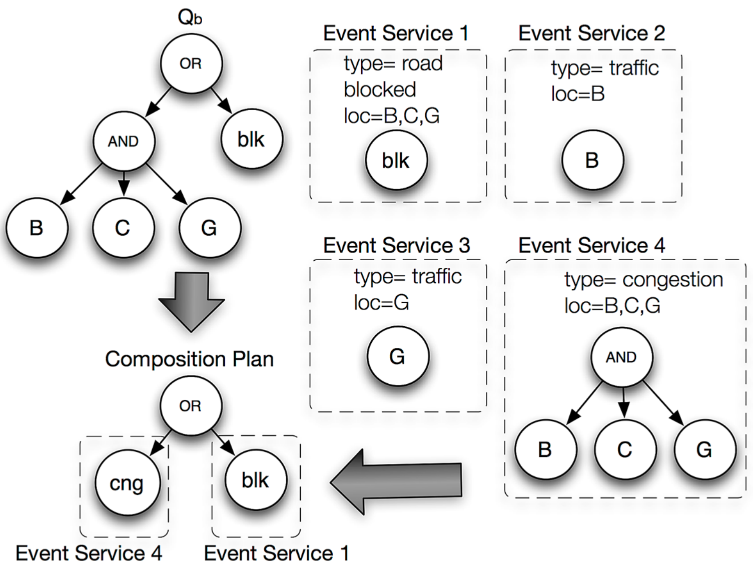
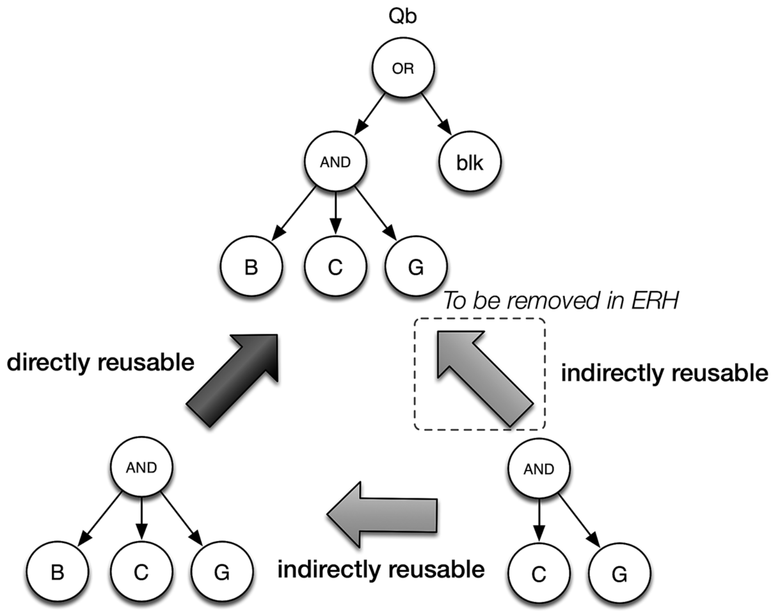


Figure 5. Example of event pattern reusability



Given a set of event services and the reusable relations among them, an Event Reusability Hierarchy (ERH) can be constructed as an index for CESs. An ERH is a Directed-Acyclic-Graph (DAG). Its nodes represent canonical event patterns of event services and its edges represent reusable relations between nodes. Constructing an ERH requires iteratively inserting canonical event patterns into the hierarchy. If not all nodes can be inserted to a single ERH, we obtain a set of separated ERHs, called an Event Reusability Forest (ERF). ERH and ERF are used to accelerate event service composition, as detailed in (F. Gao, Curry, & Bhiri, 2014).

4. QOS MODEL AND AGGREGATION SCHEMA

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.

4.1. QoS Properties of Event Services

As a proof-of-concept, we do not intend to create a complete and precise QoS ontology for event services. We refer readers to (Iggena et al., 2014) for a more comprehensive study on the Quality Ontology[7] for IoT-enable event streams. In this paper, we consider the following QoS attributes from (Iggena et al., 2014) are relevant for QoS propagation and aggregation, including:

- *Latency (L)* 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);
- *Price (P)* describes the monetary cost for an event service;
- *Energy Consumption (Eng)* describes the energy cost for an event service;
- *Network Consumption (Net)* describes the usage of network (measured by messages consumed per unit time) for an event service, fewer messages consumed will also reduce the burden of event engines while evaluating event patterns;
- *Availability (Ava)* describes the possibility of an event service being accessible, it can be numerically represented in percentages;
- *Completeness (C)* describes the completeness of events delivered by an event service, it can be numerically represented as recall rates in percentages;
- *Accuracy (Acc)* describes the possibility of getting correct event messages, it can be numerically represented in percentages; and
- *Security (S)* describes the security protocol used by event services numerically represented as integer security levels (higher numerical value indicates higher security levels).

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

4.2. Quailty of Service Aggregation

The QoS performance of an event service composition is influenced by three factors: *Service Infrastructure*, *Composition Pattern,* and *Event Engine*. The *Service Infrastructure* refers to computational hardware, service Input/Output (I/O) implementation and the physical network connection; it determines the inherent I/O performance 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 QoS performance varies depending on which services are used to produce member events and how event operators correlate them. 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 2 summarizes how the different QoS parameters of an event service composition are calculated based on these three factors. In this table, we assume event engines are free to use, always ready to accept new queries when deployed, and do not introduce security issues. When network consumption is measured in the number of event messages, we consider it irrelevant to the service infrastructure (connection types, messaging formats etc.). We do consider the effect of service

Table 2. Overall quailty of service calculation

Dimensions	QoS Symbols			Overall QoS 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	-	$P = P_i + P_c$
Energy	Eng_i	Eng_c	Eng_e	$Eng = Eng_i + Eng_c + Eng_e$
Network Consumption	-	Net_c	-	$Net = Net_c$
Availability	Ava_i	Ava_c	-	$Ava = Ava_i \times Ava_c$
Completeness	C_i	C_c	C_e	$C = C_i \times C_c \times C_e$
Accuracy	Acc_i	Acc_c	Acc_e	$Acc = Acc_i \times Acc_c \times Acc_e$
Security	S_i	S_c	-	$S = min(S_i, S_c)$

infrastructure over accuracy because out-of-synchronization messages during data transmission may lead to incorrect results. Also, note that for a simple event service that is not equipped with CEP engines (e.g., a sensor service), its overall quality vector is identical to the quality vector of the *Service Infrastructure*.

The *Composition Pattern* (EST of a composed event pattern) is a key factor in aggregating QoS properties for event service compositions. A step-wise aggregation over ESTs is used to aggregate QoS properties. More specifically, we apply aggregation rules iteratively from leaves to roots on ESTs. Aggregation rules for different QoS dimensions can be event operator dependent or independent, i.e., the aggregated QoS on a parent node in an EST may or may not depend on the event operator type of the parent node. Table 3 shows the detailed rules for each quality dimension with regard to different composition patterns. 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., the price can be charged over subscription time or volume, similar for energy consumption. For simplicity we assume they are specified over time. The overall price and energy cost of $\mathscr{E}$ (denoted $P_c(\mathscr{E})$ and $Eng_c(\mathscr{E})$, similar for other QoS dimensions) is the sum of the price and energy cost of the *Immediately Composed Event* services (denoted $\mathscr{E}_{ice}$), i.e., leaves in the EST of $\mathscr{E}$;
2. **Network Consumption** is an operator independent property. The aggregated network consumption is the sum of the product of the completeness and the frequencies of the services in $\mathscr{E}_{ice}$ (denoted $f(e)$, $e \in \mathscr{E}_{ice}$). We refer readers to (Gao, Curry, & Bhiri, 2014) for detailed descriptions on estimating frequencies of event services;
3. **Availability**, **Accuracy** and **Security** are operator independent properties. The availability and accuracy of $\mathscr{E}$ is the product of event service availability and accuracy in $\mathscr{E}_{ice}$. The rationale of aggregating accuracy with multiplication is that we consider a result is incorrect if one of the inputs used to calculate the result is incorrect. By the same logic, we aggregated the availability with multiplication. The security level is determined by the most vulnerable event service in $\mathscr{E}_{ice}$;
4. **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 a sequence or repetition, the latency

Table 3. Quailty of service aggregation rules based on composition patterns

QoS Dimensions for Event Service E	Aggregation Rules	Applicable Event Operators
$P_c(\mathscr{E})$	$\sum_{e \in \mathcal{E}_{ice}} P_c(e)$	*And, Or, Sequence, Repetition*
$Eng_c(\mathscr{E})$	$\sum_{e \in \mathcal{E}_{ice}} Eng_c(e)$	*And, Or, Sequence, Repetition*
$Net_c(\mathscr{E})$	$\sum_{e \in \mathcal{E}_{ice}} C_c(e) \cdot f(e)$	*And, Or, Sequence, Repetition*
$Ava_c(\mathscr{E})$	$\prod_{e \in \mathcal{E}_{ice}} Ava_c(e)$	*And, Or, Sequence, Repetition*
$Acc_c(\mathscr{E})$	$\prod_{e \in \mathcal{E}_{ice}} Acc_c(e)$	*And, Or, Sequence, Repetition*
$S_c(\mathscr{E})$	$min\left\{S_c(e) \mid e \in \mathcal{E}_{ice}\right\}$	*And, Or, Sequence, Repetition*
$L_c(\mathscr{E})$	$L_c(e), e\ is\ the\ last\ event\ in\ \mathcal{E}_{dse}$	*Sequence, Repetition*
	$avg\left\{L_c(e) \mid e \in \mathcal{E}_{dse}\right\}$	*And, Or*
$C_c(\mathscr{E})$	$\frac{min\left\{C_c(e) \cdot f(e) \mid e \in \mathcal{E}_{dse}\right\}}{card(\mathcal{E}) \cdot f(\mathcal{E})}$	*And, Sequence, Repetition*
	$\frac{max\left\{C_c(e) \cdot f(e) \mid e \in \mathcal{E}_{dse}\right\}}{card(\mathcal{E}) \cdot f(\mathcal{E})}$	*Or*

of $\mathscr{E}$ is same as the last event in the Direct Sub-Events of $\mathscr{E}$ (denoted $\mathscr{E}_{\text{dse}}$), i.e., sub-events whose root nodes are child nodes of the root of $\mathscr{E}$. Since it is hard to predict when the last direct sub-event occurs under parallel operators (i.e., *And* and *Or* operator), we make an approximation with the average of the latencies of the event services in $\mathscr{E}_{\text{dse}}$;

5. **Completeness** is an operator dependent property. The completeness of $\mathscr{E}$ can be estimated based on its direct sub-event frequencies ($f(e)$, $e \in \mathscr{E}_{\text{dse}}$), and completeness ($C_c(e)$, $e \in \mathscr{E}_{\text{dse}}$):

$$\frac{min\left\{C_c(e) \cdot f(e) \mid e \in \mathcal{E}_{dse}\right\}}{card(\mathcal{E})} \text{ or } \frac{max\left\{C_c(e) \cdot f(e) \mid e \in \mathcal{E}_{dse}\right\}}{card(\mathcal{E}) \cdot f(\mathcal{E})}$$

represents how often all or any direct sub-event instances would occur under the influences of the completeness, where *card($\mathcal{E}$)* gives the cardinality[8] of the root operator of $\mathcal{E}$. By dividing this frequency with the estimated frequency of $\mathcal{E}$, we derive the completeness of $\mathcal{E}$.

4.3. Event QoS Utility Function

In order to choose the best service composition under users' QoS constraints and preferences, a QoS utility function is needed. While defining a sophisticated utility function is not the focus of this paper, we use the Simple Additive Weighting (SAW (Zeleny & Cochrane, 1982)) technique to define the service QoS utility. SAW is widely used in QoS-aware service composition optimization, e.g., in (Alrifai & Risse, 2009), (Wu et al., 2013). It is worth noting that by applying SAW we have the following three assumptions (Rowe & Pierce, 1982):

- **Risk independence:** Implying the uncertainty of QoS values are not considered, i.e., the probability p of a QoS attribute q has the value v is not modelled;
- **Preferential independence:** Implying preferences over values of a set of QoS attributes do not depend on the values of other attributes; and
- **Utility independence:** Implying QoS attributes are independent of each other.

The first assumption is trivial for us because we do not take into account the probability of errors in the QoS aggregation. The second assumption also holds because a total order can be applied to all eight QoS dimensions, regardless of the values in other dimensions. For example, under any circumstances, it is safe to assume that lower latency is more desirable, as well as higher accuracy. The third assumption is a simplification of the real-world settings: sensors *may* use more energy to take more samples in order to achieve higher accuracy, and the network consumption of a composition plan *may* have a correlation with the completeness of the composition. In the current QoS model, we do not consider the correlation between quality attributes.

Given a quality vector of an event service composition $Q = <L, P, Eng, Net, Ava, C, Acc, S>$ representing the service QoS capability, we denote q as one of the eight quality dimensions in the vector, $O(q)$ as the theoretical optimum value (e.g., for latency the optimum value is 0 seconds) in the quality dimension of q, $C(q)$ as the user-defined value specifying the hard constraints (i.e., worst acceptable value, e.g., 1 second for latency) on the dimension, and $0 \le W(q) \le 1$ as the weighting function of the quality metric, representing users' preferences (e.g., W(L)=1 means latency is highly important for the user and W(L)=0 means latency is irrelevant for the user). Furthermore, we distinguish between QoS properties with positive or negative tendency: $Q=Q_{+} \cup Q_{-}$, where $Q_{+}=\{Ava, C, Acc, S\}$ is the set of properties with the positive tendency (larger values the better), and $Q_{-}=\{L, P, Eng, Net\}$ is the set of properties with the negative tendency (smaller values the better). The QoS utility U is derived by:

$$U = \sum_{q_i \in Q_+} \frac{W(q_i)\cdot(q_i - C(q_i))}{O(q_i) - C(q_i)} - \sum_{q_j \in Q_-} \frac{W(q_j)\cdot(q_j - O(q_j))}{C(q_j) - O(q_j)}$$

According to the above equation, the best event service composition should have the maximum utility U. A normalised utility with values between [0,1] can be derived using the function:

$$\bar{U} = (U + |Q_-|)/(|Q_+| + |Q_-|)$$

5. 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 levels of granularity. Each member event detection task can be achieved by subscribing to a set of event services. In large-scale scenarios, it is highly inefficient to enumerate all possible compositions of event services and evaluate their overall performance. In this section, we propose a heuristic method based on *Genetic Algorithms* (GA) to derive near-optimal event service compositions efficiently. The algorithm is intended to be deployed as an event service discovery/composition engine on a centralized server.

Typically, GAs require 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 mutation until termination conditions are met. The GA approach in this section follows these steps. The "fitness" of each solution can be evaluated by the QoS utility function in Section 4.3.

5.1. Population Initialization

Given an event request with a requested canonical event pattern *ep*, 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 an ERF.

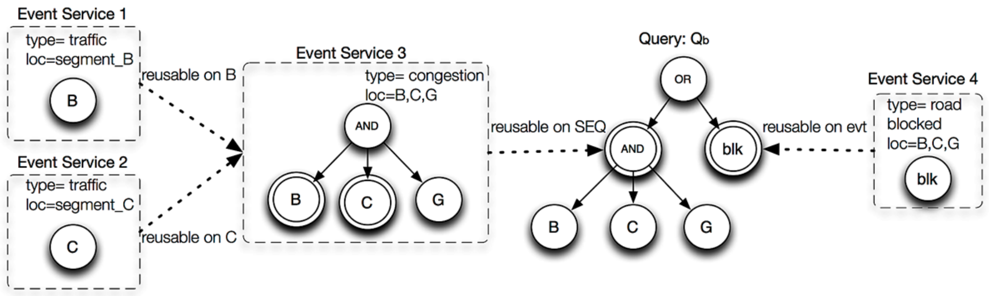
When an event pattern *ep* is inserted into the ERF, we can mark its *reusable nodes* denoted as depicted in Figure 6. Intuitively, a reusable node $n \in N_r$ for *ep* means the sub-tree of the EST of *ep* with *n* as the root can be replaced by another EST in the service repository. Evidently, *ep* has at least one ACP, which is the canonical event pattern in the event request. This ACP implies all event operators of *ep* are evaluated locally by the event engine that the composition plan is deployed on and subscriptions are made only to PESs. If one or more nodes in N_r are event operators, more ACPs of *ep* can be derived by collapsing different combinations of the reusable operator nodes, i.e., subscribing to relevant CESs and allowing the event operators in *ep* to be evaluated externally.

It is worth noting 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. The reusable relations can be efficiently retrieved from the ERF. Therefore, the enumeration of ACPs can be done efficiently.

5.2. Genetic Encodings for Concrete Composition Plans

Individuals (CCPs) in the population need to be genetically encoded to represent their various characteristics (composition patterns). In a typical encoding for Web Service compositions, each

Figure 6. Marking the reusable nodes



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 the service task to which they relate. Similarly, we encode the event detection tasks (leaf nodes) in a 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 to, since the CCPs are partially ordered 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 counterparts. Therefore, global identifiers are assigned to all the nodes in the CCPs and a leaf node in a CCP is encoded with a string of node identifiers as a prefix representing the path of its ancestors and a service identifier indicating service binding for the leaf, as shown in Figure 7. For example, a gene for the leaf node "n13" in P_2 is encoded as a string with prefix "n10n11" and a service id for the traffic service candidate for road segment B, i.e., "es3", hence the full encoding of n13 is <n10n11,es3>. The complete set of encodings for every gene constitutes the chromosome of P_2.

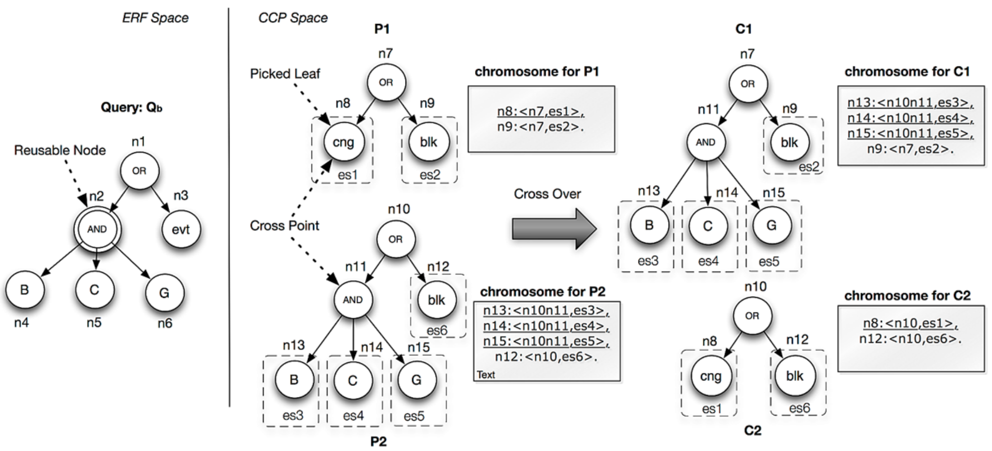
5.3. Crossover and Mutation Operations

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

5.3.1. 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 7. 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:

Figure 7. Example of genetic encodings and crossover



1. Pick randomly a leaf node l_1 from P_1, create the node type prefix ntp_1 from the genetic encoding of P_1, i.e., $code_1$, as follows: replace each node id in the prefix of $code_1$ with the operator type;
2. For each leaf l_1 in P_2, create the node type prefix ntp_2 from $code_2$ (i.e., encodings for l_2) and compare it with ntp_1. If $ntp_1 = ntp_2$ and the event semantics of l_1 and l_2 are equivalent, i.e., they are merged into the same node in the *ERF*, then mark l_1, l_2 as the crossover points n_1, n_2. If $ntp_1 = ntp_2$ but the pattern of l_1 is reusable to l_2 or l_2 is reusable to l_1, then search back on $code_1$, $code_2$ until the cross points n_1, n_2 are found on $code_1$, $code_2$ such that $T(n_1) \doteq T(n_2)$, i.e., the sub-patterns of P_1,P_2 with n_1, n_2 as the root node of the ESTs of the sub-patterns are semantically equivalent;
3. If ntp_1 is an extension of ntp_2, e.g., $ntp_1 = (And;Or;Seq)$, $ntp_2 = (And;Or)$ and the pattern of l_1 is reusable to l_2 in the *ERF*, then search back on $code_1$ and try to find n_1 such that the sub-pattern with EST $T(n_1)$ is equivalent to l_2. If such n_1 is found, mark l_2 as n_2;
4. If ntp_2 is an extension of ntp_1, do the same as step 3 and try to find the cross point n_2 in $code_2$;
5. Whenever the cross points n_1, n_2 are marked in the previous steps, stop the iteration. If n_1 or n_2 is the root node, return P_1, P_2 as they were. 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.

5.3.2. Mutation

The mutation operation changes the composition plan for a leaf node in a CCP. To do that we first select a random leaf node in a CCP. If the leaf is a PES, then the mutation is simply changing the service binding to a different PES. Otherwise, we treat the event pattern of the leaf node as a new event request that needs to be composed, then we use the same process specified in the population initialization to create a random composition plan for the leaf and replace the leaf node in the original CCP with the composition plan.

5.3.3. Elitism

We use an *Elitism* method in the GA. More specifically, after the selection in every generation, we add an exact copy of the best individual directly into the next generation without crossover or mutation (the original instance may still participate in the crossover and mutation). Elitism ensures the best individual is kept through the evolution until a better individual has occurred.

6. EVALUATION

In this section, we present the evaluation results of the proposed approaches. We put our experiments in the context of the travel-planning scenario in Section 2 by using both real and synthetic sensor datasets for the city of Aarhus. The evaluation has two parts: in the first part, we analyse in detail the performance of the genetic algorithm. In the second part, we demonstrate the usefulness of the QoS aggregation rules. All experiments are carried out on a machine with a 2.53 GHz duo core CPU and 4 GB 1067 MHz memory. Experiment results are the average of 30 iterations.

6.1. Part 1: Performance of the Genetic Algorithm

In this part of the evaluation, we compare the QoS utility derived by Brute-Force (BF) enumeration and the developed GA. Then, we test the scalability of the GA. Finally, we analyse the impact of different GA parameters and provide guidelines to identify optimal GA parameter settings.

6.1.1. Datasets

Open Data Aarhus (ODAA)[9] is a public platform that publishes sensor data and metadata about the city of Aarhus. Currently there are 449 pairs of traffic sensors in ODAA. Each pair is deployed on one street segment for one direction and reports the traffic conditions on the street segment. These traffic sensors are used in the experiments to answer requests on travel planning. We also include some other sensors in our dataset that might be used in traffic monitoring and travel planning, e.g., air pollution sensors and weather sensors. These sensors are not actually relevant to requests like Alice's (i.e., Q_a in Figure 3) or Bob's (i.e., Q_b in Figure 4), i.e., they are noise to queries like Q_a and Q_b (but could be used in other travel related queries). In total we use 900 real sensors from ODAA, in which about half of them are noise. We denote this dataset sensor repository R_0.

Each sensor in R_0 is annotated with a simulated random quality vector $<L, Acc, C, S>$ where $L \in$ [0ms, 300ms], Acc, $C \in$[50%,100%], $S \in$[1,5] and frequency $f \in$[0.2Hz,1Hz]. We do not model price or energy consumption in the experiments because their aggregation rules are similar to network consumption. For similar reasons we also do not model availability. To test the algorithms on a larger scale, we further increase the size of the sensor repository by adding N functionally equivalent sensors to each sensor in R_0 with a random quality vector, resulting in the 9 different repositories as shown in Table 4. In the experiments we use a loose constraint[10] to enlarge the search space and we set all QoS weights set to 1.0[11]. The queries used in the experiments are summarised in Table 5.

6.1.2. QoS Utility Results and Scalability

In this set of experiments, we first demonstrate the usefulness of the GA by comparing it to a BF algorithm and a random pick approach. Figure 8 shows the experimental results for composing Q_a over R_3 to R_9 (R_1 and R_2 are not tested here because their solution spaces are too small for GA), where Q_a has 6 service nodes and 1 operator. A more complicated variant of Q_a with 8 service nodes and 4 operators is also tested, denoted Q_a'.

The best utility obtained by the GA is the highest utility of the individual in the last generation before the GA stops. In the current implementation, the GA is stopped when the current population size is less than 5 or the difference between the best and the average utility in the generation is less than 0.01, i.e., the evolution has converged. Given the best utility from BF $\bar{U}_{bf}$, best utility from GA $\bar{U}_{ga}$ and the random utility of the dataset $\bar{U}_{rand}$}, we calculate the degree of optimization as $d = \left(\bar{U}_{ga} - \bar{U}_{rand}\right) / \left(\bar{U}_{bf} - \bar{U}_{rand}\right)$. From the results in Figure 8 we can see that the average d= 89.35% for Q_a and Q_a'. In some cases, the BF algorithm fails to complete, e.g., Q_a over R_8 and R_9, because of memory limits (heap size set to 1024 MB). We can see that for smaller repositories, d is bigger. This is because under the same GA settings[12], the GA has a higher chance of finding the global optimum during the evolution when the solution space is small and the *elitism* method described in

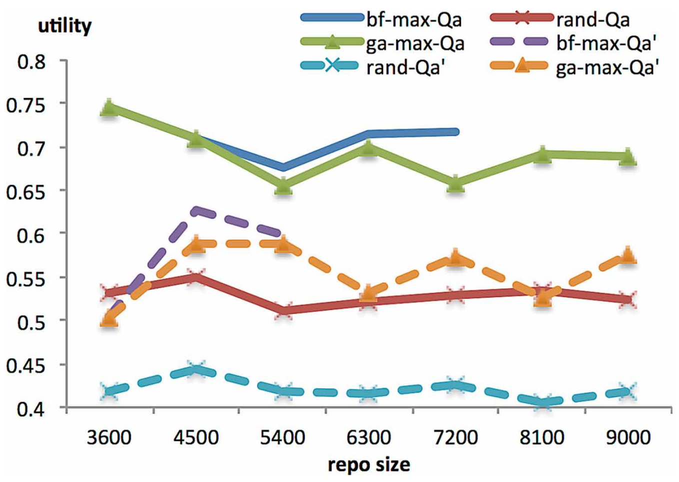
Table 4. Simulated sensor repositories

	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9
N	1	2	3	4	5	6	7	8	9
Total Size	1800	2700	3600	4500	5400	6300	7200	8100	9000

Table 5. Queries used in experiments

Query	Description	Nodes
Q_a	Alice's query on estimated travel time on route (Figure 3).	1 AND, 6 streams.
Q_b	Bob's query on traffic condition, (Figure 4).	1 AND, 1 OR, 4 streams.
Q_a'	A variants of Q_a with more nodes.	1 AND, 3 random operators, 8 streams.
Q_b'	A variant of Q_b with more nodes.	1 AND, 1 OR, 10 streams

Figure 8. QoS utilities of BF, GA and random pick



Section 5.3 makes sure that, if found, the global optimum "survives" till the end of evolution, e.g., in the GA results for Q_a over R_3 and R_4 in Figure 8.

It is evident that a BF approach for QoS optimization is not scalable because of the NP-hard nature of the problem. We analyse the scalability of the GA using different repository sizes, query sizes (total number of event operator nodes and event service nodes in the query), as well as different number of CESs in the ERF.

From the results in Figure 9 we can see that the composition time of Q_a grows linearly for GA when the size of the repository increases. To test the GA performance with different query sizes using different operators, we use the EST of Q_b as a base and replace its leaf nodes with randomly created sub-trees (invalid ESTs excluded). Then we test the GA convergence time of these queries over R_5. Results from Figure 10 indicate that the GA execution time increases linearly with regard to the query size.

Figure 9. GA scalability over repository size

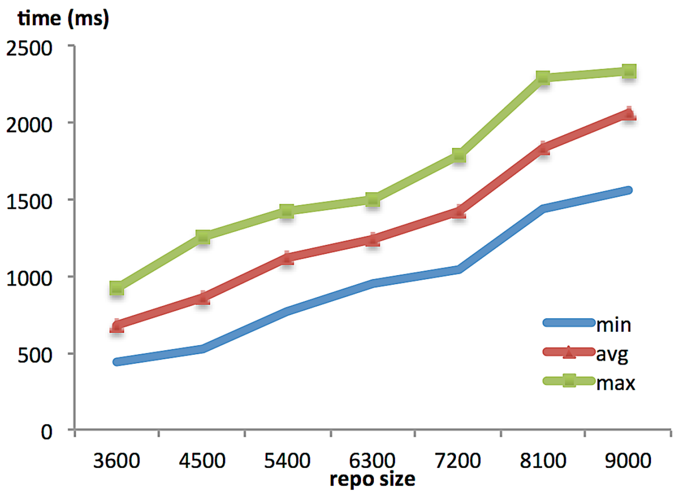
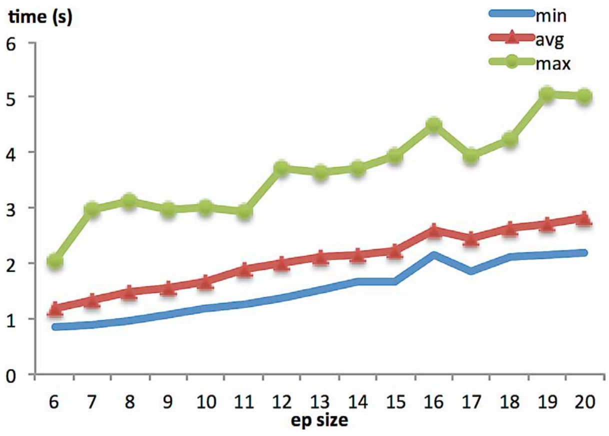


Figure 10. GA scalability over EP size

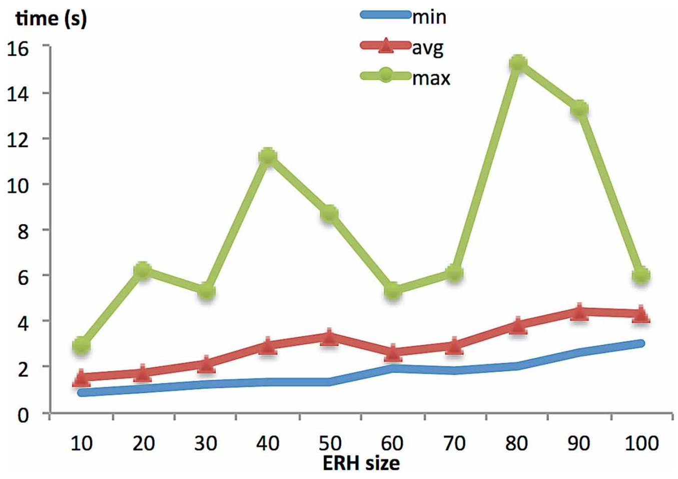


In order to test the scalability over different number of CESs in the ERF (called ERF size), we deploy 10 to 100 random CESs to R_5, resulting in 10 new repositories. We test the GA on a query created in the previous step (denoted Q_b') with the size of 12 nodes (2 operators, 10 sensor services) and record the execution time in Figure 11. To ensure each CES could be used in the composition plan, all CESs added are sub-patterns of Q_b'. From the results we can see that although the increment of the average execution time is generally linear, in some rare test instances there are "spikes", such as the maximum execution time for ERF of size 40 and 80. After analysing the results of those cases, we found that most (over 90%) of the time is spent on population initialisation, and this is caused by the complexity of the ERF, i.e., number of edges considered during ACP creation.

6.1.3. Fine-Tuning the Parameters

In the experiments above, a fixed set of settings is used as the GA parameters, including crossover rate, mutation rate and population size. In order to find good settings of the GA in our given problem domain, we fine tune the mutation rate, population size and crossover rate based on the default setting used above, i.e., we change one parameter value at a time while keeping other parameters unchanged.

Figure 11. GA scalability over ERH size



In order to determine the effect of the parameter tuning, we define a Cost-Effectiveness score (i.e., CE-score) as follows: given the random pick utility of a dataset $\bar{U}_{rand}$, we have the final utility derived by GA $\bar{U}_{ga}$ and the number of milliseconds taken for the GA to converge t_{ga}, CE-score = $(\bar{U}_{ga}-\bar{U}_{rand})*10^5/t_{ga}$. We test two queries Q_a, Q_b' over two new repositories R_5', R_9', which are R_5 and R_9 with 50 and 100 additional CESs, respectively. Hence we have 4 test combinations on both simple and complex queries and repositories. The results of fine-tuning the mutation rate, population size and crossover rate are shown in Figure 12, Figure 13 and Figure 14.

From the results in Figure 12 we can see that the optimal mutation rate is quite small for all tests, i.e., from 0% to 0.4%. Results in Figure 13 indicate that for smaller solutions spaces such as Q_a over R_5' and R_9', the optimal initial population size is smaller, i.e., with 60 individuals in the initial population. For more complicated queries and larger repositories, using a larger population size e.g., 100, is more cost-efficient. Results from Figure 14 indicate that for Q_a over R_5', the optimal crossover rate is 35% because the global optimum is easier to achieve and more crossover operations

Figure 12. CE-score over mutation rate

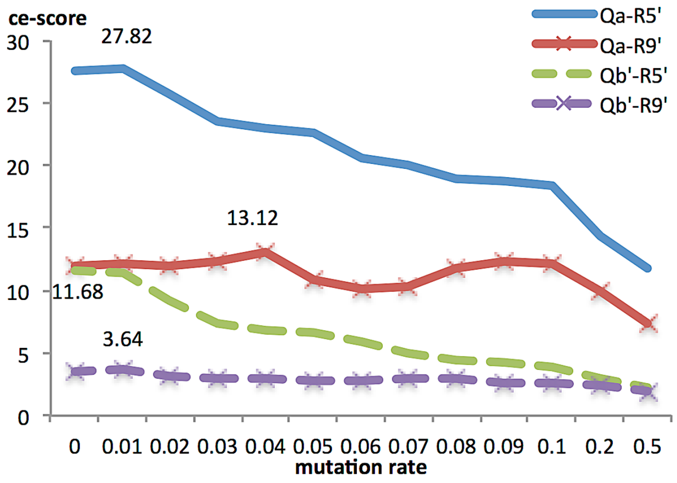


Figure 13. CE-score over population size

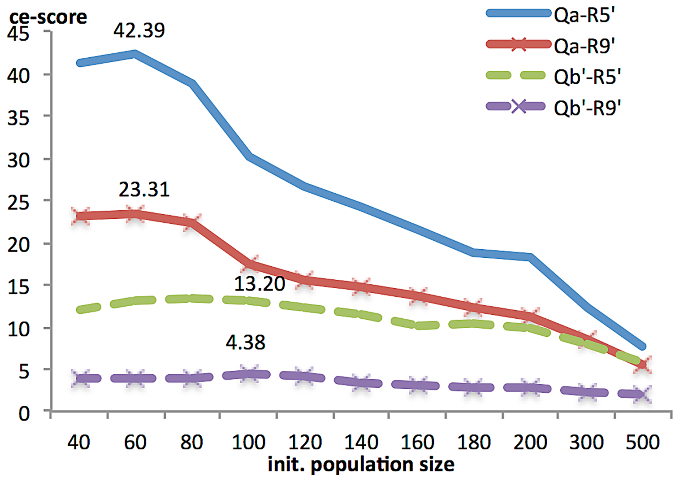
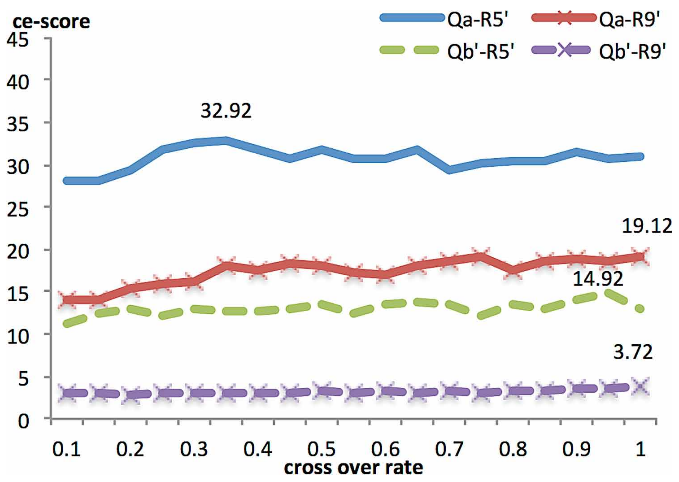


Figure 14. CE-score using over crossover rate

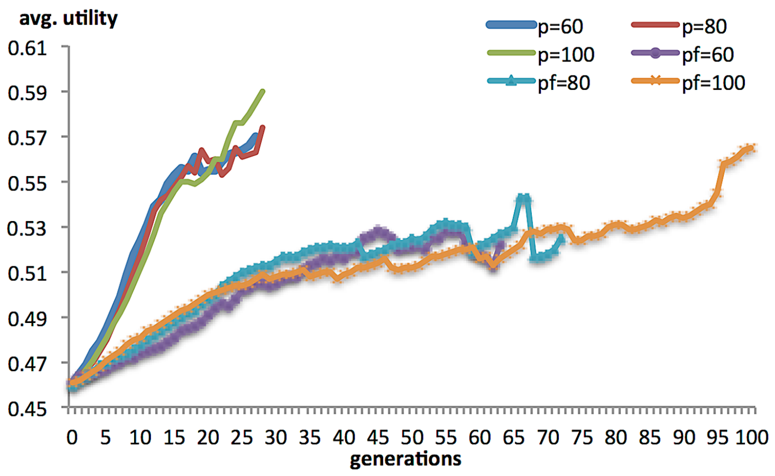


bring overhead. However, for more complicated queries and repositories, a higher crossover rate, e.g., from 90% to 100%, is desired. It is worth noticing that in the results from Figure 13 and Figure 14, the changes of the score for Q_b' over R_9' is not significant. This is due to the fact that the GA spends much more time trying to initiate the population, making the cost-effectiveness score small and the differences moderate.

In the previous experiments, we use the selection policy such that every individual is chosen to reproduce once (except for the elite whose copy is also in the next generation). This will ensure the population will get smaller as the evolution progresses and the GA will converge quickly. This is desirable in our case because the algorithm is executed at run-time and is time sensitive. However, it is also possible to allow an individual to reproduce multiple times and keep a fixed population size during the evolution.

In order to compare the differences of having a fixed or flexible population size, we show the average utility (of Q_b' over R_9') over the generations in Figure 15. The results show that the number of generations for flexible population sizes is similar while larger sizes achieve higher utilities. In

Figure 15. Average utility using flexible ("p=x") and fixed ("pf=x") population size



addition, the duration of generations in fixed population sizes is very different: for a fixed population size of 60 the GA converges in about 60 generations and for the size of 100 it lasts more than 100 generations. Larger sizes also produce better final results in a fixed population, but it is much slower and the utilities are lower than those obtained from flexible populations. In summary, we can confirm that using a flexible population size is better than a fixed population size for the GA described in this section.

6.2. Part 2: Validation of QoS Aggregation Rules

In this part of evaluation, we show how the QoS aggregation works in a simulated environment.

6.2.1. Datasets and Experiment Settings

In order to demonstrate the effect of QoS aggregation and optimisation, we generate two composition plans CP_1 and CP_2 with the GA for Q_a over R_9' using same constraints specified in 0. CP_1 is optimized for latency, with the weight of latency set to 1.0 and other QoS weights set to 0.1; while CP_2 is optimized for network consumption, with the weight of network consumption set to 1.0 and others 0.1. The reason we generate two plans for optimizing latency and network consumption is that the resulting plans are the very different in structure, as shown in Figure 16.

When the two composition plans are generated, we transform the composition plans into stream reasoning queries (e.g., C-SPARQL query). We evaluate the queries over the traffic data streams produced by ODAA sensors. According to the composition plan and the event service descriptions involved in the plans, we simulate the QoS of the event services on a local test machine, i.e., we create artificial delays, wrong and lost messages according to the QoS specifications in event service descriptions, and set the sensor update frequency as the frequency annotated (so as to affect the messages consumed by the query engine). Security is annotated but not simulated, because the aggregation rule for security is trivial, i.e., estimated to be the lowest security level. Notice that the simulated quality is the *Service Infrastructure* quality. We observe the results and the query performance over these simulated streams and compare it with the QoS estimation using the rules in Table 2 and Table 3, to see the differences between the practical quality of the composed event streams and the theoretical quality as per our estimation.

6.2.2. Simulation Results

The results of the comparison between the theoretical and simulated quality of the event service composition is shown in Table 6. The first column is the quality dimensions of the two composition plans, the second column is the computed quality values based on the aggregation rules defined in Table 3. These rules take into account the *Composition Pattern* of the query as well as the *Service Infrastructure* quality of the composed services. We denote this quality QoS_{cp}. However, this is not the end-to-end QoS, because the quality of the event stream engine needs to be considered. To get the stream engine performance we deploy the queries with optimal *Service Infrastructure* quality, i.e.,

Figure 16. Composition plans for Q_a under different weight vectors

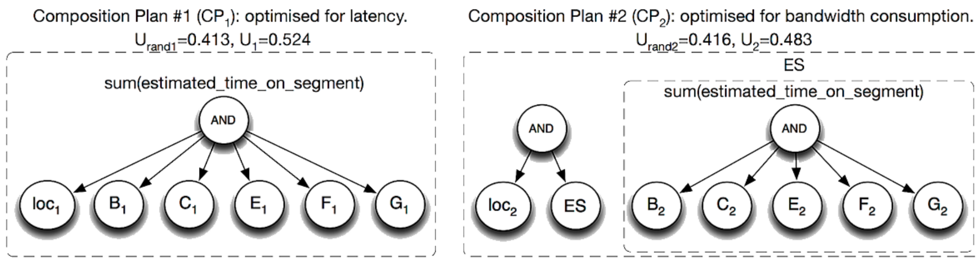

Composition Plan #1 (CP_1): optimised for latency.
U_{rand1}=0.413, U_1=0.524
sum(estimated_time_on_segment)
AND
loc_1
B_1
C_1
E_1
F_1
G_1
Composition Plan #2 (CP_2): optimised for bandwidth consumption.
U_{rand2}=0.416, U_2=0.483
ES
sum(estimated_time_on_segment)
AND
loc_2
ES
AND
B_2
C_2
E_2
F_2
G_2

Table 6. Validation for QoS aggregation and estimation

	Composition Pattern	Event Engine	End-to-End Simulated	End-to-eEnd Deviations
Plan 1 (CP_1)				
Latency	40 ms	604 ms	673 ms	+4.50%
Accuracy	50.04%	100%	51.43%	+2.78%
Completeness	87.99%	97.62%	72.71%	-14.89%
Network Consumption	4.05 msg/s	4.05 msg/s	3.84 msg/s	-5.19%
Plan 2 (CP_2)				
Latency	280 ms	1852 ms	2328 ms	+9.19%
Accuracy	53.10%	100%	51.09%	-3.79%
Completeness	87.82%	73.18%	46.31%	-17.96%
Network Consumption	0.37 msg/s	0.40 msg/s	0.32 msg/s	-13.51%

no artificial delay, mistake or missing events, and we record the quality of query executions in the third column. We denote this engine quality QoS_{ee}. The simulated end-to-end quality is recorded in the fourth column, denoted QoS_s. We calculate the theoretical end-to-end quality based on QoS_{cp} and QoS_{ee} using Table 2. Notice that the *Service Infrastructure* qualities of the queries themselves are not considered, since we do not measure the results provided to external service consumers, rather, the quality measurement stops at the point when query results are generated. We denote this theoretical end-to-end quality QoS_t and calculate the deviation $d = (QoS_s/QoS_t) - 1$, which is recorded in the last column. From the results we can see that the GA is very effective in optimizing latency for CP_1 and network consumption for CP_2: the latency of the former is 1/7 of the latter and event messages consumed by the latter are less than 1/8 of the former.

We can also see that the deviations of latency and accuracy are moderate for both plans. However, the completeness estimation is about 15% to 18% different to the actual completeness. For the network consumption in CP_1 the estimation is quite accurate, i.e., about 5% more than the actual consumption. However, the network consumption for CP_2 deviates from the estimated value by about 13.51%. The difference is caused by the unexpected drop in C-SPARQL query completeness when a CES with imperfect completeness is reused in CP_2, which suggests that an accurate completeness estimation of a service could help improving the estimation of the network consumption for event service compositions using the service.

Another interesting observation from Table 6 is that the end-to-end delay of CP_2 is about 1650 ms longer than CP_1, while the artificial delay imposed by the CES is no more than 280 ms. This is caused by the internal query mechanism of the C-SPARQL engine: when composed queries are registered to the same engine instance, the engine will take much more time to process the query because of the concurrency. This suggests that a federated manner of query composition over distributed engine instances is desirable.

7. 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 (Jaeger, Rojec-Goldmann, & Muhl, 2004) and (Wu et al., 2013). In this paper, we extract some typical QoS properties from the existing work and define a similar utility function based on SAW. However, the aggregation rules in existing works focus on conventional web services rather than complex event services, which need a different QoS aggregation schema. For example, the event engine 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 (Jaeger et al., 2004)).

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 Genetic Algorithm based approaches are developed, e.g., in (Zhang & Li, 2004), (Zhang, Su, & Chen, 2006), (Gao, Cai, & Chen, 2007) and (Karatas & Kesdogan, 2013). In (Zhang & Li, 2004) the chromosomes are encoded with binary bits representing whether a service is selected or not. The problem with this approach is that the readability of the genomes is poor and the chromosome length is not fixed during evolution. In (Zhang et al., 2006) a two-dimensional genome encoding is proposed to express all execution paths while considering task relations, but its crossover and mutation need validation. In (Gao et al., 2007), the authors use tree coding chromosomes, crossovers operate on sub-trees and mutations operate on leaf nodes to avoid invalid reproductions. In (Karatas & Kesdogan, 2013) 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 with service tasks on different granularity levels. A more recent work in (Wu et al., 2013) 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 (Wu et al., 2013) indicate that up to 10% utility enhancement can be obtained by expanding the search space. Composing events on different granularity levels is also a desired feature for CES composition. However, (Wu et al., 2013) only caters for *Input*, *Output*, *Precondition* and *Effect* (IOPE) based service compositions. Complex event service composition requires a pattern-based reuse mechanism (Gao, Curry, & Bhiri, 2014). As a result, it requires different genetic encoding mechanisms and crossover operations.

8. CONCLUSION AND FUTURE WORK

In this paper, we address the issue of enabling QoS-aware event stream federation and optimisation using services in an IoT context. A QoS aggregation schema is proposed to calculate the overall QoS vector for a federated IoT stream (an event service composition). Based on user-defined constraints and preferences, a QoS utility function is defined to calculate the degree of optimisation for event composition. A genetic algorithm for creating optimal event service compositions is developed and evaluated. We evaluate the proposed approach over a travel-planning scenario with both real and synthetic datasets. The experimental results show that the genetic algorithm is scalable, and can give near-optimal (about 89% optimal) results efficiently. We provide analysis on how to fine-tune the GA parameters including mutation rate, crossover rate and population size in order to achieve the best performance of GA. The experiments on the validation of QoS aggregation show that our estimation model does not deviate too far from the practical results.

As future work, we plan to optimise the GA approach with regard to the population initialisation, because we found that this process takes the majority of the computation time. We also plan to investigate how interdependent quality metrics can be modelled and optimised. Moreover, investigating the QoS aggregation over different RDF stream processing engines is also on the agenda.

REFERENCES

Alrifai, M., & Risse, T. (2009). Combining global optimization with local selection for efficient QoS-aware service composition.*Proceedings of the 18th international conference on World wide web* (pp. 881–890). doi:10.1145/1526709.1526828

Berbner, R., Spahn, M., Repp, N., Heckmann, O., & Steinmetz, R. (2006). Heuristics for qos-aware web service composition. *Proceedings of the International Conference on Web Services ICWS'06* (pp. 72–82). doi:10.1109/ICWS.2006.69

Etzion, O., & Niblett, P. (2010). *Event processing in action*. Manning Publications Co.

Gao, C., Cai, M., & Chen, H. (2007). QoS-aware Service Composition Based on Tree-Coded Genetic Algorithm. *Proceedings of the 31st Annual International Computer Software and Applications Conference* (Vol. 1, pp. 361–367). Washington, DC, USA: IEEE Computer Society. doi:10.1109/COMPSAC.2007.174

Gao, F., Curry, E., Ali, M., Bhiri, S., & Mileo, A. (2014). QoS-aware Complex Event Service Composition and Optimization using Genetic Algorithms.*Proceedings of the 12th International Conference on Service Oriented Computing*, Paris, France.Springer. doi:10.1007/978-3-662-45391-9_28

Gao, F., Curry, E., & Bhiri, S. (2014). Complex Event Service Provision and Composition based on Event Pattern Matchmaking.*Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems,*Mumbai, India. ACM. doi:10.1145/2611286.2611287

Hasan, S., & Curry, E. (2014). Thematic Event Processing. In ACM/IFIP/USENIX Middleware conference 2014. doi:10.1145/2663165.2663335

Hinze, A., Sachs, K., & Buchmann, A. (2009). Event-based applications and enabling technologies.*Proceedings of the Third ACM International Conference on Distributed Event-Based Systems* (pp. 1–15). New York, NY, USA: ACM. doi:10.1145/1619258.1619260

Iggena, T., Kümper, D., & Tönjes, R. (2014, May). Kontinuierliche Bewertung von Informationsqualität in Stream-basierten Smart City Architekturen. ITG-Fachbericht-Mobilkommunikation–Technologien und Anwendungen, Osnabrück, Germany.

Jaeger, M. C., Rojec-Goldmann, G., & Muhl, G. (2004). QoS aggregation for Web service composition using workflow patterns.*Proceedings of the Eighth IEEE InternationalEnterprise Distributed Object Computing Conference. EDOC 04* (pp. 149–159). doi:10.1109/EDOC.2004.1342512

Karatas, F., & Kesdogan, D. (2013). An Approach for Compliance-Aware Service selection with genetic algorithms. In *Service-Oriented Computing* (pp. 465–473). Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-45005-1_35

Mela, K., Tiainen, T., & Heinisuo, M. (2012). Comparative study of multiple criteria decision making methods for building design. *Advanced Engineering Informatics*, *26*(4), 716–726. doi:10.1016/j.aei.2012.03.001

Rowe, M. D., & Pierce, B. L. (1982). Sensitivity of the weighting summation decision method to incorrect application. *Socio-Economic Planning Sciences*, *16*(4), 173–177. doi:10.1016/0038-0121(82)90036-2

Wu, Q., Zhu, Q., & Jian, X. (2013). QoS-Aware Multi-granularity Service Composition Based on Generalized Component Services. In S. Basu, C. Pautasso, L. Zhang, & X. Fu (Eds.), *Service-Oriented Computing* (Vol. 8274, pp. 446–455). Springer Berlin Heidelberg; doi:10.1007/978-3-642-45005-1_33

Zeleny, M., & Cochrane, J. L. (1982). *Multiple criteria decision making* (Vol. 25). McGraw-Hill New York.

Zeng, L., Benatallah, B., Ngu, A. H. H., Dumas, M., Kalagnanam, J., & Chang, H. (2004). QoS-aware middleware for Web services composition. *IEEE Transactions on* Software Engineering, *30*(5), 311–327. doi:10.1109/TSE.2004.11

Zhang, C., Su, S., & Chen, J. (2006). A novel genetic algorithm for qos-aware web services selection. In *Data Engineering Issues in E-Commerce and Services* (pp. 224–235). Springer. doi:10.1007/11780397_18

Zhang, L.-J., & Li, B. (2004). Requirements Driven Dynamic Services Composition for Web Services and Grid Solutions. *Journal of Grid Computing*, *2*(2), 121–140. doi:10.1007/s10723-004-4202-1

ENDNOTES

1 CityPulse 101 scenarios: http://www.ict-citypulse.eu/scenarios/
2 Event Processing Technical Society - Glossary of Terms: http://www.complexevents.com/wp-content/uploads/2011/08/EPTS_Event_Processing_Glossary_v2.pdf
3 The Open Group's definition for service: https://www.opengroup.org/soa/source-book/soa/soa.htm
4 Complex Event Service Ontology: http://citypulse.insight-centre.org/ontology/ces/
5 OWL-S: Semantic Markup for Web Services: http://www.w3.org/Submission/OWL-S/
6 In the ESTs depicted in this paper, event operator nodes are labelled with operator types and a location update event is labelled "loc", traffic report events are labelled as the street segments in the map (See Figure 1) where the sensors are deployed
7 Quality Ontology: https://mobcom.ecs.hs-osnabrueck.de/cp_quality/
8 The function card($\mathscr{E}$) gives the cardinality of the root operator of event pattern in $\mathscr{E}$, the cardinality of a repetition node is greater than 1, for other operators the cardinality is 1.
9 Open Data Aarhus: Error! Hyperlink reference not valid.
10 Constraint used in the evaluation: (L>= 3000ms, Acc>= 0, C >= 0, S >= 1, B <= 50)
11 In this paper, we consider the weights representing users' personal preferences and do not differentiate between "good" or "bad" weight settings.
12 In the experiments in Section 6.1, the GA has the following parameter settings: the initial population size is set to 200, crossover rate is 95%, mutation rate is 3%.

Feng Gao received his BSc degree in Software Engineering from Wuhan University, China, in September 2008. He received an MEng degree in Telecommunication from Dublin City University with honors in 2009. Currenty he is a PhD student at the INSIGHT Centre for Data Analytics, National University of Ireland, Galway. His current research interests include Semantic Web, Complex Event Processing and Service Computing. He has passed his PhD viva recently on 24th March 2016 and is expected to be coferred with PhD from National University of Ireland, Galway in June 2016.

Muhammad Intizar Ali is an Adjunct Lecturer, Research Fellow and Project Leader at the Unit for Reasoning and Querying at Insight Centre for Data Analytics, National University of Ireland, Galway. His research interests include Semantic Web, Data Integration, Internet of Things (IoT), Linked Data, Federated Query Processing, Stream Query Processing and Optimal Query Processing for large scale distributed data sources. He is actively involved in various EU funded and industry-funded projects aimed at providing IoT-enabled adaptive intelligence for smart city applications and smart enterprise communication systems. He is serving as a PC member of various journals, international conferences and workshops. He is also actively participating in W3C efforts for standardization in RDF Stream Processing Community Group. Dr. Ali obtained his PhD (with distinction) from Vienna University of Technology, Austria in 2011.

Edward Curry a research leader at the Insight Centre for Data Analytics (www.insight-centre.org) and a funded investigator at LERO The Irish Software Research Centre (www.lero.ie). Edward has worked extensively with industry and government advising on the adoption patterns, practicalities, and benefits of new technologies. Edward has published over 120 scientific articles in journals, books, and international conferences. He has presented at numerous events and has given invited talks at Berkeley, Stanford, and MIT. In 2010, he was a guest speaker at the MIT Sloan CIO Symposium to an audience of 600+ CIOs and senior IT executives. His research projects include studies of smart cities, energy intelligence, semantic information management, event-based systems, and collaborative data management. He is a member of the scientific leadership committee of Insight, and a Lecturer in Informatics at the National University of Ireland Galway (NUIG). He is the Vice President of the Big Data Value Association (www.BDVA.eu) a non-profit industry-led organisation with the objective of increasing the competitiveness of European Companies with data-driven innovation.

*Alessandra Mileo is a Senior Research Fellow, Adjunct Lecturer and Unit Leader at the INSIGHT Research Center for Data Analytics, NUI Galway, Ireland (formerly DERI). She holds an MSc (2002) and a PhD (2006) in Computer Science from the University of Milan. Since 2011, she has been leading the Reasoning and Querying Unit, focusing on the ability to unlock the potentials hidden in the fast growing torrent of data generated on the Internet of Things, and investigating the resulting economical and social impact on application domains including Smart Cities, Smart Transport and remote health monitoring. She has been involved in the organization of numerous conferences and symposia, and she is an active PC member of more than 40 conferences and high impact Journals. Dr. Mileo is a Steering Committee member of the Web Reasoning and Rule Systems Association (RRA), member of the Association of Logic Programming (ALP) and area editor for the ALP Newsletter (Database and Semantic Web area), Research Committee Member in INSIGHT@NUIG, member of the Italian AI Association (AI*IA), the Italian working group on Knowledge Representation and Automated Reasoning (RCRA) and the ARCOE association as a chair of the ARCOE workshop series. She is Principal Investigator of the EU FP7 CityPulse project on large-scale data analytics for smart cities, and for the primary industry collaboration within the Research Centre portfolio on Enabling the Internet of Everything: a Linked Data infrastructure for networking, managing and analyzing streaming information. Her research interests include web stream reasoning, deductive systems, Internet of Things, Semantic Web and Linked Data, adaptive algorithms, inductive learning, large-scale query processing and federation, context-aware systems, and she has published more than 40 articles in international conferences and Journals. Dr. Mileo has proven experience in supervising both undergrad and postgrad students in Computer Science and Digital Humanities around her topics of expertise and research interests. In relation to standardization activities, Dr. Mileo is an active member of the W3C RDF Stream Processing and Web of Things groups. As a Principal Investigator in the EU FP7 project CityPulse, she is also involved in the IERC Activiy Chain AC7 - Cognitive Technologies for IoT. As part of the Steering Committee of the INSIGHT Centre for Data Analytics, Dr. Mileo contributed to the global strategy for H2020 which includes the proposal of a Magna Carta for Data.*