# QoS-aware Adaptation for Complex Event Service*

Feng Gao
feng.gao@insight-centre.org

Muhammad Intizar Ali
ali.intizar@insight-centre.org

Edward Curry
edward.curry@insight-centre.org

Alessandra Mileo
alessandra.mileo@insight-centre.org

Insight Centre for Data Analytics
National University of Ireland, Galway

## ABSTRACT

Smart City applications often use event processing techniques to detect coarse-grained events and situations from fine-grained events of the physical and social world. They operate in dynamic environments in which the properties of underlying resources and streams need to be constantly updated according to changes and events in the real world (e.g. sensor readings, network availability, weather conditions, and temperature). In most of the existing solutions matchmaking between the requirements expressed by event consumers and available event providers is carried out at design-time. This approach is often far from optimal and its deficiencies become even more obvious in smart city scenarios due to their inherently dynamic stream properties. In this paper we discuss a solution for quality-aware adaptive complex event processing using a service-oriented approach. We detail the automatic adaption strategies and evaluate them in a smart city scenario with both real and synthesised datasets.

## CCS Concepts

**•Applied computing → Service-oriented architectures; Event-driven architectures; •Computer systems organization → Dependable and fault-tolerant systems and networks;**

## Keywords

Service Oriented Architecture; Complex Event Processing; Service Adaptation; Quality-of-Service

*(Produces the permission block, and copyright information). For use with SIG-ALTERNATE.CLS. Supported by ACM.



*SAC 2016,April 04-08, 2016, Pisa, Italy*

 ISBN 978-1-4503-3739-7/16/04...$15.00
DOI: http://dx.doi.org/10.1145/2851613.2851806

## 1. INTRODUCTION

Complex Event Processing (CEP) has been widely discussed and used for deductive reasoning over dynamic data streams to detect complex situations according to predefined event patterns [15]. During the past decade, research efforts in Event Processing Network (EPN) aim to carry out CEP in a distributed fashion [7]. However, advancements in Internet-of-Things (IoT) and Smart City applications bring new challenges to existing EPNs, e.g., incorporating heterogeneous data interfaces, data semantics and CEP platforms [3]. Previous works in [9, 10] aim to address these challenges by providing CEP capabilities as Complex Event Services (CESs), which describe the event patterns in their service descriptions and allow CEP capabilities to be reused within different CEP platforms. A network of CESs and Simple Event Services[1] (SESs) constitute an Event Service Network (ESN), as depicted in Figure 1. In [8] an Automatic Complex Event Implementation System (ACEIS) is proposed as a middleware for managing the modelling, composition and implementation of CESs. However, it did not address the problem of dynamic service adaptation under quality changes.

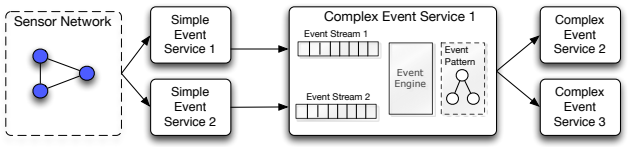
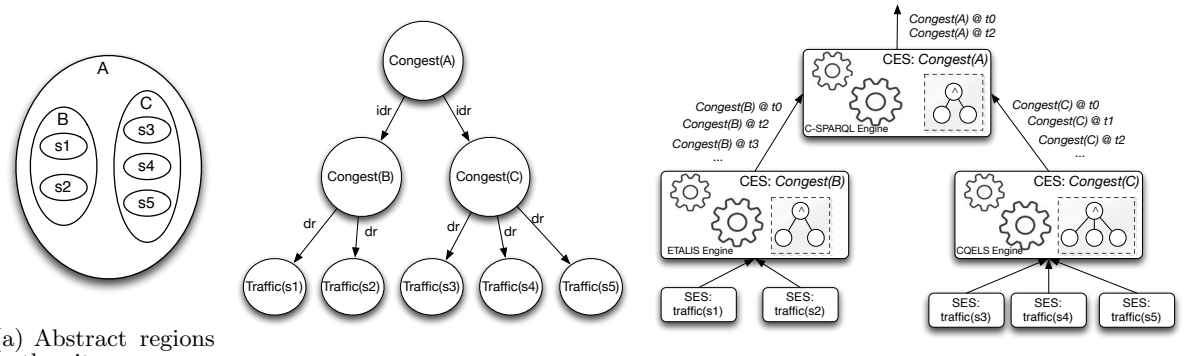



Figure 1: Overview of an event service network

Adaptability is important for service-oriented systems [6]. Adaptability is crucial especially in the context of IoT, because IoT service qualities are prone to environmental changes [11]. For example, the accuracy of a sensor might be affected by its battery level [4], air temperature, humidity [12] etc. ACEIS should have the ability to automatically detect service failures or constraint violations according to users' requirements at run-time and make appropriate adjustments, including re-compose and re-deploy CESs, to adapt to changes.

[1]An SES is an event service that may deliver simple or complex events but no event patterns are described in the service description, hence can only be identified by event types and attributes.




(a) Abstract regions in the city

(b) ERH for traffic congestion monitoring event services

(c) ESN for traffic congestion monitoring event services

Figure 2: Example of an ESN for traffic monitoring

In this paper we extend the framework in [8] and provide means for Quality-of-Service (QoS) aware adaptive event processing over IoT data/event streams. The contribution of this paper is summarised as follows:

* We extend the ACEIS architecture in [8] to enable QoS-aware CES adaptation and elaborate on the details of three different adaptation strategies.
* We evaluate the proposed adaptation techniques with a prototype implementation using both real and simulated urban datasets.

The remainder of this paper is organized as follows: Section 2 introduces a motivation scenario in urban traffic monitoring; Section 3 presents the extended ACEIS architecture and elaborates in details how the QoS-aware adaptations are made; Section 4 presents and discusses evaluation results for different adaptation strategies; Section 5 describes the state-of-the-art on QoS-aware adaptive service composition before we summarise the outcome and future steps in Section 6.

## 2. MOTIVATION SCENARIO

Figure 2 shows a simplified example of federated CESs. Suppose 3 CESs are deployed to monitor the traffic congestion events in different regions of a city. As shown in Figure 2(a), region $A$ consists of region $B$ and $C$, i.e., $A = B \cup C$, while $B$ and $C$ consist of several different streets, i.e., $B = (s1, s2), C = (s3, s4, s5)$. The congestion event in region $A$ is detected when both $B$ and $C$ are congested, i.e., $Congest(A) := Congest(B) \wedge Congest(C)$, while $Congest(B)$ and $Congest(C)$ are detected based on the traffic reports produced by the traffic sensors deployed on the relevant streets, i.e., primitive events $Traffic(s_n)$. Domain experts can define different rules for the congestion event, e.g., average vehicle speed below a threhold or vehicle count per road distance is higher than a threhold etc. The ESN topology containing the relevant CESs and SESs created by ACEIS is shown in Figure 2(c). Notice that $Congest(A)$ is evaluated by a C-SPARQL engine [2], and it reuses event detection results from $Congest(B)$ and $Congest(C)$, which are evaluated by ETALIS [1] and CQELS [14] engines, respectively. This example demonstrates how event processing capability can be provided in a platform independent manner, in which different event engines can be used, as long as all patterns are described in a mutually accepted ontology, such as the CES ontology[2]. By analysing the event patterns evaluated by the CESs, an *Event Reusability Hierarchy* (ERH) can be constructed using the method provided in [10], as shown in Figure 2(b). An ERH describes how event services can directly or in-directly reuse (denoted *dr* and *idr* in Figure 2(b)) other services by comparing the semantics of the events they provide. An ERH serves as an index for the event services and accelerates the event service composition [9].

When composing the CES for detecting $Congest(A)$, a user may specify QoS constraints (expressed as QoS value thresholds) and preferences (expressed as weights over QoS metrics) on the composition plan created. A composition plan describes how existing services are reused in a CES composition, the reused services are referred to as memeber event services in the composition plan. ACEIS is capable of satisfying the constraints during CES composition and rank the composition plans based on the preferences. The created composition plan may use different combinations of the CESs and SESs in Figure 2. At run-time, when an event service involved in the composition plan has a QoS update, two questions arise: 1) will the quality update significantly affect the QoS of the CES and result in violation of the constraints and 2) if yes, how to modify the current composition plan and satisfy the constraints?

## 3. EVENT SERVICE ADAPTATION

Existing approaches to data stream processing address different issues related to streams and data processing such as stream data management, query processing, and data mining. However, it is still an open challenge to properly address issues that are more closely related to the quality of a federated stream, more precisely integration of the federated data/event streams while keeping their quality metrics in mind. To facilitate adaptability in quality-aware federation of IoT streams for smart city applications, the following

[2]CES Ontology: citypulse.insight-centre.org/ontology/ces

actions have to be considered:

- **Monitor Quality Updates:** to monitor any updates in the quality metrics of the IoT streams involved in stream federation,
- **Evaluate Criticality:** to determine whether any particular quality update is critical and if there is any adaptation action that should be carried out, and
- **Perform Adaptation:** once a critical update is confirmed, perform necessary actions for automatic adaptation according to the new conditions/environment.

In the following we first present the architecture of ACEIS with a focus on the above actions are implemented in the adaptation module. Then, we elaborate the strategies for creating new service compositions at runtime, i.e., performing adaptation to keep the constraints satisfied with an abstract example.

## 3.1 ACEIS Architecture

ACEIS is developed as a middleware to model, discover, compose and adapt CESs. Figure 3 illustrates the overall architecture of ACEIS. ACEIS consists of three main components: *Application Interface*, *Semantic Annotation* and *ACEIS Core* component.

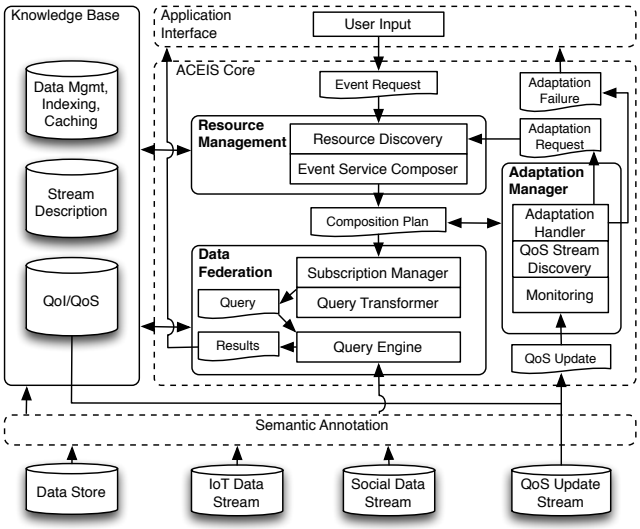



Figure 3: ACEIS architecture overview (extended from [8])

The *Application Interface* interacts with end users as well as the ACEIS core modules. It allows users to provide inputs required by the application and presents the results to the users. Event requests are generated from user inputs and sent to the *ACEIS Core* for further processing. The *Semantic Annotation* component annotates syntactical information with semantic terms. It annotates both static information (e.g., event service descriptions with CES ontology) and dynamic information (e.g., IoT messages and sensor observations with the SSN[3] ontology), so that a semantic discovery of event services and a semantic querying and reasoning over event messages can be realised.

The *ACEIS Core* module serves as a middleware between low level event streams and upper level applications and business processes. The ACEIS core consists of three major components: resource management, data federation and adaptation manager. The *Resource Management* is responsible for performing pattern-based [9] and QoS-aware [10] discovery and composition of CESs. The *Data Federation* is responsible for implementing the composition plan over the ESN. It uses a subscription manager to subscribe to the relevant semantic event streams in the composition plan and transform the composition plan into a semantic stream query in order to detect the requested complex events.

The *Adaptation Manager* comprises of three main modules, namely:

**QoS Stream Discovery:** this module finds the relevant quality update streams to subscribe to for the composition plan.

**Monitoring:** this module subscribes to the relevant QoS streams and continuously monitors and verifies whether quality scores of all the contributing data streams in a composition plan are compliant to the event request. We adopt the QoS aggregation method described in [9] for estimating the overall QoS performance of a composition plan and determining whether a QoS constraint is violated.

**Adaptation Handler:** this module is triggered if any of the user defined QoS constraints and preferences are violated. It utilizes different adaptation strategies and tries to determine the scope of the adaptation. If an adaptation is possible, it invokes the *Resource Management* component to find replacements for parts of (or the entire) original composition plan. Then it creates the new composition plan via merging or replacing the original composition plan and send it back to the *Data Federation* component for query redeployment and other subsequent actions. In order to avoid conflicting adaptations, no parallel adaptations are allowed, i.e., when an adaptation is in action, all QoS updates are ignored.

## 3.2 Adaptation Strategies

We distinguish between 3 different adaption strategies for creating new composition plans: *local*, *global*, and *incremental* adaptations. All strategies query the ERH to find valid candidate services. In the following we elaborate on these 3 strategies using an example ERH shown in Figure 4.

Recall that in Section 2, an ERH is informally described as a hierarchy that captures reusability between event services. Now we formally define an ERH as a directed-acyclic-graph (DAG) consisting of a set of nodes and edges, i.e., $erh = (V, R)$ where $V$ is a set of event services and $R \subset V \times V$ is a set of binary relations over nodes. We distinguish between two types of relations: *reuse* and *equivalent* relation in $R$, i.e., $R = R_r \cup R_e$ where $R_r$ and $R_e$ represent *reuse* and *equivalent* relations, respectively. $R_r$ is a transitive, asymmetric relation such that if $(v_1, v_2) \in R_r$, the semantics of the events provided by $v_2$ is covered by those of $v_1$, i.e., whenever an event instance $e_1$ is detected by $v_1$, there

[3]Semantic Sensor Network ontology: http://www.w3.org/2005/Incubator/ssn/ssnx/ssn

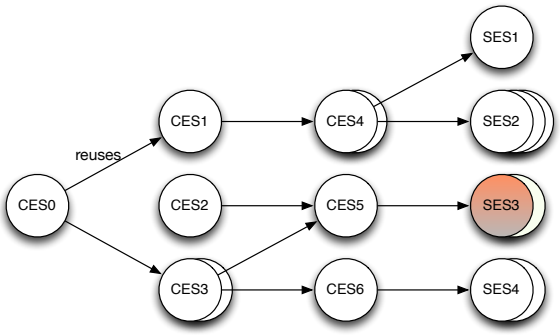



Figure 4: Example of an ERH

should be at least one $e_2$ detected by $v_2$, where $e_2$ belongs to the *Event Instance Sequence* (EIS) that triggers $e_1$. $R_e$ is a transitive, symmetric and reflexive relation such that if $(v_1, v_2) \in R_e$, the semantics of the events provided by $v_1$ is equivalent to those of $v_2$, i.e., whenever an event instance $e_1$ is detected by $v_1$, there should be exactly one event instance $e_2$ detected by $v_2$ such that $e_1$ and $e_2$ are triggered by the same EIS. An ERH is constructed once when an ACEIS server initializes, and constantly maintained by adding and removing services in the ERH, until the server is shut down.

In Figure 4 a (partial) example of an ERH (denoted $erh = (V, R)$) is presented, which contains 11 functionally different CES and SES nodes. Arrows in Figure 4 represent the $R_r$ relations, and $R_e$ relations are abstracted as stacked nodes, representing functional identical services. Suppose an instance of the ACEIS adaptation manager is currently monitoring the service $CES_0$, and its current composition plan contains a set of member event services $mes(CES_0) = \{CES_4, SES_3, CES_6\}$, the adaptation manager for $CES_0$ will subscribe to the QoS updates for all event services in $mes(CES_0)$. When a QoS update, say, for $SES_3$ is detected, it will recalculate the aggregated QoS metrics for $CES_0$ to see if it still complies with the user-defined constraint $C$. If the constraint $C$ still holds for $CES_0$, it will do nothing except to update the QoS for $SES_3$ in the composition plan of $CES_0$ and publish a QoS update for $CES_0$ to all other interested adaptation manager instances. Otherwise it marks $SES_3$ as a *critical node* (marked in red), triggers an adaptation process, and tries to create a new composition plan for $CES_0$ that satisfies $C$ using one of the following three strategies:

- **local** adaptation that finds all functional equivalent services to $SES_3$, ranks them based on constraint $C$ and preference $P$, and then substitutes $SES_3$ with the highest ranking replacement $SES_3'$ in the current composition plan of $CES_0$;
- **global** adaptation that recomposes a new composition plan entirely for $CES_0$ based on $C, P$ and
- **incremental** adaptation that follows the steps:
  1. try *local* adaptation, if failed i.e., no substitutes available or the substitution of $SES_3$ cannot satisfy $C$ then,
  2. try to recompose the critical node, if failed, i.e., no composition possible, e.g., the *critical node* here is an SES, or replacing the *critical node* with the new composition cannot satisfy $C$ then,
  3. find all Intermediate Nodes (denoted $IN$) between $CES_0$ to $SES_3$ in $erh$ such that
  $$IN \subseteq V \mid \forall v \in IN \implies (v, SES_3) \in R_r^*$$
  where $R_r^*$ is the transitive closure of $R_r \subseteq R$,
  4. starting from the node with shortest to the largest distance to $SES_3$ in $IN$, i.e., in the sequence of $CES_5$, $CES_3$ and $CES_0$, mark the node as the *critical node* and repeat step 1 and 2 on this node until a satisfying new composition plan is created and
  5. if all above steps failed to create a valid composition plan, exit with an adaptation failure notification to the users or the application, triggering the necessary recovery mechanisms.

**Algorithm 1** CES adaptation algorithm

**Require:** Composition Plan: $comp$, QoS update: $qUpdate$, Constraint: $C$, Adaptation Mode: $mode$, Event Reusability Hierarchy: $erh$
**Ensure:** Adapted Composition Plan: $resultPlan$
**procedure** ADAPT($comp, qUpdate, C, mode, erh$)
  $needAdpt \leftarrow$ CHECKCONST($comp, qUpdate, C$)
  $resultPlan \leftarrow \emptyset$
  **if** $needAdpt = true$ **then**
    **if** $mode = local$ **then**
      $resultPlan \leftarrow$ LOCALADPT($comp, qUpdate, C$)
    **else if** $mode = global$ **then**
      $resultPlan \leftarrow$ GLOBALADPT($comp, qUpdate, C$)
    **else if** $mode = incremental$ **then**
      $resultPlan \leftarrow$ INCREMENTALADPT($comp, qUpdate, C, erh$)
    **end if**
  **end if**
  **return** $resultPlan$
**end procedure**
**Require:** Composition Plan: $comp$, QoS update: $qUpdate$, Constraint: $C$, Event Reusability Hierarchy: $erh$
**Ensure:** Adapted Composition Plan: $resultPlan$
**procedure** INCREMENTALADPT($comp, qUpdate, C, erh$)
  $resultPlan \leftarrow \emptyset$
  $IN \leftarrow$ GETIN($comp, erh$) $\cup$ $comp$
  **for** $criticalService \in IN$ **do**
    $resultPlan \leftarrow$ LOCALADPT($comp, qUpdate, C$)
    **if** $resultPlan = \emptyset$ **then**
      $subP \leftarrow$ GETSUBPATTERN($comp, criticalService$)
      $subResult \leftarrow$ GLOBALADPT($subP, qUpdate, C$)
      $resultPlan \leftarrow$ MERGERESULT($comp, subResult$)
      **if** CHECKCONST($resultPlan, qUpdate, C$) **then**
        $result \leftarrow \emptyset$
      **end if**
    **end if**
    **if** $resultPlan \neq \emptyset$ **then**
      $break$
    **end if**
  **end for**
  **return** $resultPlan$
**end procedure**

Algorithm 1 provides the pseudo code for the adaptation algorithm, with a focus on incremental adaptation. Intuitively, each of the above three strategies has its own merits (and drawbacks). *Local* adaptation causes the smallest changes to the composition plan and requires the least computation effort, however it has a relatively low chance of a successful adaptation, and even if it succeeds, the resulting new event service composition may have a low overal

QoS, since the substitute options are limited. *Global* adaptation ensures a high probability of success while it gets the best possible (with regard to the composition algorithm used) resulting QoS. However, it may change dramatically the structure of the original service composition and requires the same time of composing the original service in the service planning phase, which might take too long for a adjustment during run-time. *Incremental* adaptation is a more "balanced" choice between *local* and *global* adaptation, i.e., it changes the scope of adaptation when necessary, thus on average, it takes the intermediate time to adapt and creates the intermediate resulting QoS. Notice that in the case when an *incremental* adaptation is regressed into a *global* one, they produce the same quality results, but the *incremental* approach may take even more time than the *global*, due to the overhead of failed attempts. In Section 4 the above intuitions are verified with experiments.

## 3.3 Adaptation for Service Failures

It is worth mentioning that by adopting the QoS aggregation methods described in [9], we can handle adaptations for constraints over eight QoS metrics, including latency, accuracy, availability, completeness, security and energy/-monetary/bandwidth consumption. However, service failures, such as server offline, or connection broken, are not directly supported. Nevertheless, service failures can be easily adopted as critical QoS updates and trigger the adaptation, as long as those service failures provide explicit notifications to the service consumers. In cases where no explicit notifications are provided, prediction methods based on statistics or patterns can be used to detect service failures and trigger adaptations [17]. These methods are not detailed in this paper.

# 4. EVALUATIONS

In this section we experiment on the performance of our adaptation strategies using a traffic monitoring scenario (implemented as an extension of the example in Figure 2) in the smart city context. In the following we first present the scenario and dataset description and then show the results of the experiments as well as the analysis of the results.

## 4.1 Datasets

The city of Aarhus has deployed 449 pairs of traffic sensors on the streets to report traffic conditions. The live traffic data, along with other sensor data on air pollution, weather etc. has been made publicly available via the ODAA[4] platform. By wrapping the sensors as SESs that publish sensor data as sensor observation events, we can integrate ACEIS in the traffic monitoring scenario sketched in Figure 2.

To experiment on the adaptation capability we also collected the QoS measurements for the sensors used during August, 2014 and replay them with the sensor observations to simulate the real-time quality updates[5]. Figure 6 and 7 show the QoS analysis for an event request over the selected month.

[4]Open Data Aarhus: www.odaa.dk

[5]We are thankful for the CityPulse team in the University of Applied Sciences Osnabrück (UASO) for the QoS data assessment and collection.

In the experiments we monitor a query of traffic congestion events over a specific route in the city. Figure 5 shows the start and end locations of the queried route, which consists of 10 street segments (10 traffic sensor services deployed on the route from point A to B in Figure 5). Figure 6 shows the distribution of the accuracy measurements of the 10 sensor services during the month and Figure 7 shows the trend of the aggregated accuracy for the query during the month, i.e., for each day of the month, we observe the maximum, minimum and average of the aggregated accuracy of the query (multiplication of the accuracy of 10 sensors). From the accuracy distribution and aggregated accuracy for each day we can see that although for about 90% of the time the sensor observation is correct (100% accuracy), we still have quite some low accuracy results when we investigate large queries using observations from many sensors, which strengthens our argument on the necessity of quality-aware event service adaptation.

## 4.2 Performances of Adaptation Manager

To investigate the adaptation performance in more details, we replay the QoS updates in a random day of the month (e.g., 21st of August, 2014) and observe how the adaptation manager reacts to the quality changes during the day while monitoring the traffic condition on the route specified in Figure 5. In addition to the real-world datasets, for experimental purposes, we create synthesised datasets: for each sensor deployed in the city, we create 10 functionally equivalent virtual sensors, so that *local* adaptation is possible. QoS updates for these virtual sensors are also simulated: for each real sensor QoS update stream, we apply 10 different (and random) offsets over the timestamps (e.g., +1 hour) of the updates to create 10 virtual sensor quality updates streams. We also deploy 100 CESs in the ERH, each CES is a random combination of the street segments used in the query in Figure 5. These CESs represents traffic monitoring queries over smaller regions and their results can be reused by the investigated query, so that the *incremental* adaptation is possible.

### 4.2.1 Comparison of Different Strategies

Table 1 shows an overall comparison of different adaptation strategies. The first column lists the adaptation strategies used ("n/a" stands for no adaptation) under 3 QoS constraints: a (relatively) strict constraint requires the accuracy of query results above 90%, a medium constraint above 80% and a loose constraint above 70%. The second column lists the QoS updates that are considered critical, i.e., the updates causing constraint violations. It shows while *local* adaptation can reduce some critical quality updates, *global* and *incremental* adaptation can reduce the amount to the minimum. The reason behind this is that when an adaptation is triggered and failed, the problematic event service tends to keep causing critical QoS updates. The third column lists the number of successful adaptations. The results indicate that *local* adaptation has a much lower success rate than *global* and *incremental*. The fourth column lists the time required for the adaptations. From the results we can see that *local* adaptation is very efficient while *global* may take more than 3 seconds to complete, and *incremental* adaptation takes less time than the *global* option and more

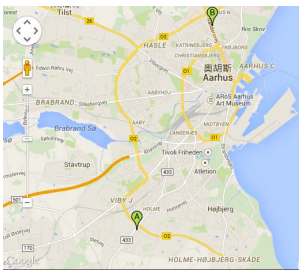



Figure 5: Traffic monitoring query on the map

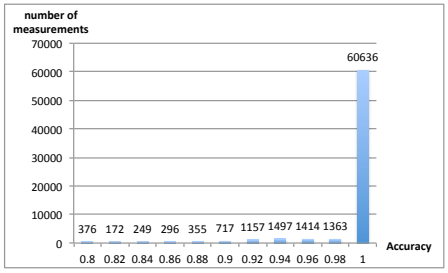



Figure 6: Accuracy distribution over a month

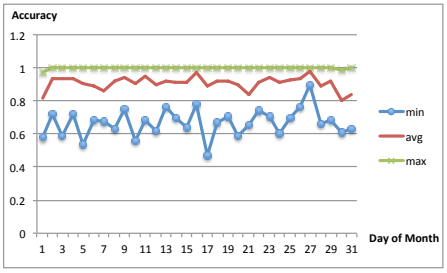



Figure 7: Accuracy trend over a month

Table 1: Comparison of adaptation strategies

| | critical updates | valid adpt. | avg. adpt. time (ms) | query results | satisfied period | total acc. changes |
|---|---|---|---|---|---|---|
| **Constraint $C_1$: accuracy > 90%** | | | | | | |
| n/a | 470 | 0 | 0 | 2160 | 22.67% | |
| local | 407 | 10 | 8 | 2145 | 22.96% | +1.18% |
| global | 17 | 17 | 3243 | 1487 | 100.00% | +43.09% |
| incremental | 16 | 16 | 2272 | 1596 | 100.00% | +44.67% |
| **Constraint $C_2$: accuracy > 80%** | | | | | | |
| n/a | 413 | 0 | 0 | 2161 | 36.32% | |
| local | 349 | 5 | 9 | 2161 | 37.53% | +1.39% |
| global | 9 | 9 | 3332 | 1654 | 100.00% | +39.47% |
| incremental | 14 | 14 | 919 | 1661 | 99.91% | +31.93% |
| **Constraint $C_3$: accuracy > 70%** | | | | | | |
| n/a | 355 | 0 | 0 | 2145 | 50.02% | |
| local | 317 | 2 | 8 | 2143 | 48.38% | -1.41% |
| global | 7 | 7 | 3446 | 1668 | 100.00% | +38.73% |
| incremental | 6 | 6 | 815 | 1836 | 100.00% | +28.15% |

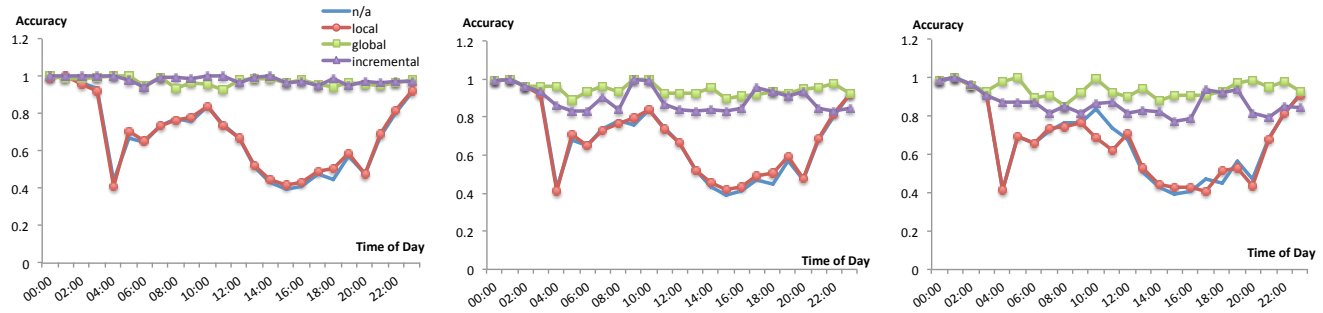
than the *local* adaptation. The efficiency and scalability of the adaptation is guaranteed by the CES composition algorithm in [9]. The fifth column lists the number of query results (i.e., congestion events) obtained from the event stream engine. If we use the "n/a" option as the baseline (i.e., assuming the event engine does not create false positive/negatives), we can see that the *global* and *incremental* adaptation suffers from high message loss ($\approx$ 30% loss in the worst case) and *local* adaptation does not lose many event messages (less than 0.7%). We will provide more analysis on this later (in Section 4.2.2). The sixth columns shows the portion of the time during the day that the constraints are satisfied using different adaptation strategies. We can see from the results that the *global* and *incremental* adaptation can always keep the constraints satisfied, while the *local* adaptation provides slightly improved satisfactory time for constraint $C_1$ and $C_2$ and has worsened the situation for constraint $C_3$. This effect is also reflected in the seventh column, where the summed accuracies of different strategies over the day are compared to the non-adapted approach.

In summary, the results in Table 1 show that the *local* adaptation is more efficient and has less message loss than the *global* and *incremental* adaptation due to the limited search space available. However, for the same reason it has a much lower success rate and quality improvement. The *local* adaptation success rate is likely to improve if there are more functional equivalent event services to adapt to. Users can choose the most suitable adaptation strategy according to their needs. To have a more intuitive representation of the accuracy trends over the day using different strategies, we plot the hourly averaged accuracy of the query in Figure 8.

### 4.2.2 *Message loss and adaptation Time*

From Table 1 we can see that the message loss is positively related to the average adaptation time. Indeed, if more time is required to make the adjustments, there is a higher chance that a query result is lost. The frequency of query result update depends on the frequency of the input streams. In the experiments above the traffic conditions are reported every 3 seconds. To see the impact of stream frequency over the message loss rate, we use *global* and *incremental* adaptation under constraint $C_2$ using different streaming intervals. The results are shown in Figure 9. From the results we can see that slowing down the streaming rate can reduce the message loss for both strategies, but it cannot eliminate them. In fact, even when we use a streaming interval of 9 seconds, the message loss is still high: $\approx$ 15% for both strategies. By further analysing the data we found out two more reasons for the message loss: 1) when a new event query is registered as a result of adaptation, a new event window is used and the previous events are discarded, thus, some query results may be lost, and 2) the semantic stream engine (e.g., C-SPARQL) takes additional time (e.g., several seconds) to get the query results after the query is registered. A possible solution would be deploying the new query along with the old one and keep the old query results until the new query is fully functional. However, it causes an overhead and we risk receiving low quality query results.

The adaptation time is an important metric while evaluating the adaptation strategies as well as deciding the best option in a particular usecase. For *local* adaptation the time required simply depends on the number of $R_e$ relations in the ERH. The *global* adaptation time depends on the efficiency of the event service composition algorithm, which is out of the scope of this paper. For *incremental* adaptation, the time required to create new composition plans largely depends on the structure and size of the ERH used. A successful *incremental* adaptation could be completed during 3 different phases in the adaptation procedure (recall the incremental adaptation steps in Section 3.2): *local replacement* (i.e., from steps 1 and 2), *parent replacement and recompose* (i.e., from steps 3 and 4 excluding global recomposition) and *global recomposition*. The *local replacement* and *global*




(a) Accuracy trend under constraint $C_1$ (b) Accuracy trend under constraint $C_2$ (c) Accuracy trend under constraint $C_3$

Figure 8: Accuracy trends under different constraints over a day using different strategies

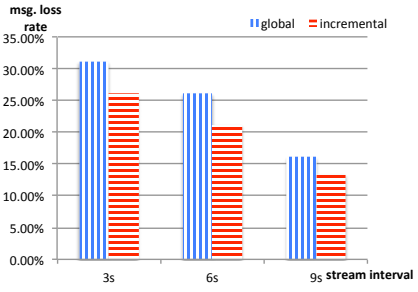



Figure 9: Message loss rate under constraint $C_2$ using different stream rate

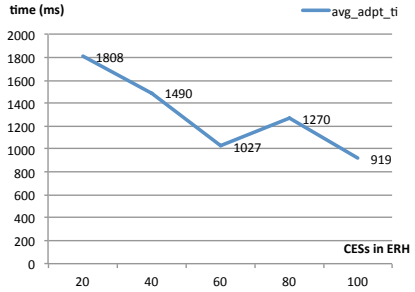



Figure 10: Avg. time used by incremental adaptation over different ERHs

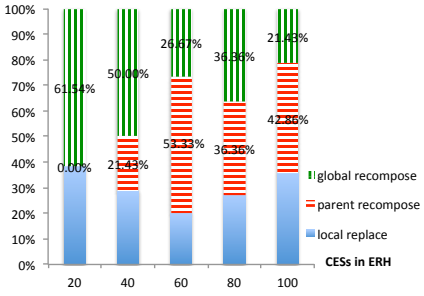



Figure 11: Distribution of incremental adaptation over different ERHs

*recomposition* take the least and most time to complete, respectively. Therefore, an ERH is ideal for *incremental* adaptation if the distribution of *global recomposition* can be minimised, i.e., it contains sufficient $R_e$ relations to enable *local replacement* as well as sufficient $R_r$ relations between the query and the critical node to enable *parent replacement*. We test the adaptation time and the distribution of successful *incremental* adaptations over ERHs with different sizes under constraint $C_2$. The results are shown in Figure 10 and Figure 11. From the results we can see that in general, the adaptation time is negatively related to the size of the ERH and is positively related to the percentage of *global recompositions* occurred.

## 5. RELATED WORKS

QoS-aware adaptive service composition and self-recovery have been discussed extensively. In [16] a multi-tier ranking system is used to categorise services based on link analysis over snapshot of the service network. This approach can select popular services in the network based on dynamic bindings, however it can only recover from service failures. Also, only service re-discovery at an atomic level is realised. In [13] ontology-based solutions are proposed to manage the service life-cycle in the cloud, including service discovery, negotiation, composition and monitoring. A fuzzy-logic-based framework is used to monitor service quality. However, it does not provide dynamic recovery mechanism. In [19] different service recovery strategies are discussed and a Dynamic Local Backup Recovery Algorithm (DLBRA) is proposed for ubiquitous services. DLBRA is quite similar to the approach in [16], only that the backups are proactively searched by the service monitor instead of taking snapshots when the recovery mechanism is triggered (as in [16]), also the quality analysis is based on a quality utility aggregated from multiple quality metrics instead of link analysis. We argue that proactively searching backups will introduce an overhead, also local recovery has a lower success rate because the choices are very limited. In [20] reinforcement learning is used to optimise service compositions, however, like [18] and other learning based approaches, the learning phase is needed. In [5] a context-aware adaptive composition over IoT services is proposed. It adopts an incremental adaptation strategy, which is similar to the idea we propose in this paper. However, it focuses on adapting contextual changes rather than quality constraint violations. Moreover, existing approaches in adaptive service computing rely on imperative workflows, which is inherently different to declarative event pattern definitions in complex event services. Therefore, they cannot be applied directly to the ESNs.

## 6. CONCLUSIONS AND FUTURE WORK

In this paper, we present a service-oriented approach for quality-aware adaptive event stream federation. We first introduce the concept of Complex Event Services (CES) and Event Service Networks (ESN). A motivation scenario in urban traffic monitoring showcases the use of ESN and raises

the questions related to QoS-aware event service adaptation. Then, we present a middleware (extended ACEIS) to manage the life-cycle of event services and discuss how QoS-aware adaptations are realised in the system. Detailed event service adaptation mechanisms are discussed, including three different adaptation strategies: *local*, *global* and *incremental* adaptation. Finally, we evaluate different adaptation strategies in the motivation scenario with both real and synthesised datasets and analyse the evaluation results. The results reveal that there is no global optimised strategy for the event service adaptation and users should choose different strategies based on their requirements as well as the characteristics of the datasets (i.e., service repositories). In future work, we plan to use more sophisticated constraint violation detection mechanisms, e.g., constraint distribution and negotiation, to expand the search space for the *incremental* adaptation. This may improve the performance of the *incremental* adaptation by increasing the distribution of *parent replacement* and reducing the adaptation time. We also plan to investigate effective methods to reduce the message loss caused by the adaptation.

## Acknowledgments

This research has been partially supported by Science Foundation Ireland (SFI) under grant No. SFI/12/RC/2289 and EU FP7 CityPulse Project under grant No.603095.

## 7. REFERENCES

[1] D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic. Ep-sparql: a unified language for event processing and stream reasoning. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 635–644, 2011.

[2] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus. C-SPARQL: a continuous query language for RDF data streams. *Int. J. Semantic Computing*, 4(1):3–25, 2010.

[3] P. Barnaghi, R. Tonjes, J. Holler, M. Hauswirth, A. Sheth, and P. Anantharam. Citypulse: Real-time iot stream processing and large-scale data analytics for smart city applications. In *ESWC*, 2014.

[4] A. Boulis, S. Ganeriwal, and M. B. Srivastava. Aggregation in sensor networks: an energy–accuracy trade-off. *Ad hoc networks*, 1(2):317–331, 2003.

[5] A. Bucchiarone, A. Marconi, M. Pistore, P. Traverso, P. Bertoli, and R. Kazhamiakin. Domain objects for continuous context-aware adaptation of service-based systems. In *Web Services (ICWS), 2013 IEEE 20th International Conference on*, pages 571–578, June 2013.

[6] E. Di Nitto, C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl. A journey to highly dynamic, self-adaptive service-based applications. *Automated Software Engineering*, 15(3-4):313–341, 2008.

[7] O. Etzion and P. Niblett. *Event processing in action*. Manning Publications Co., 2010.

[8] F. Gao, M. I. Ali, and A. Mileo. Semantic discovery and integration of urban data streams. In *5th Workshop on Semantic Smart Cities*, 2014.

[9] F. Gao, E. Curry, M. Ali, S. Bhiri, and A. Mileo. Qos-aware complex event service composition and optimization using genetic algorithms. In *Service-Oriented Computing*, volume 8831 of *Lecture Notes in Computer Science*, pages 386–393. Springer Berlin Heidelberg, 2014.

[10] F. Gao, E. Curry, and S. Bhiri. Complex event service provision and composition based on event pattern matchmaking. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, DEBS '14, pages 71–82. ACM, 2014.

[11] R. Iyer and L. Kleinrock. Qos control for sensor networks. In *Communications, 2003. ICC '03. IEEE International Conference on*, volume 1, pages 517–521 vol.1, May 2003.

[12] J. B. Johnson and G. L. Schaefer. The influence of thermal, hydrologic, and snow deformation mechanisms on snow water equivalent pressure sensor accuracy. *Hydrological Processes*, 16(18):3529–3542, 2002.

[13] K. Joshi, Y. Yesha, and T. Finin. Automating cloud services life cycle through semantic technologies. *Services Computing, IEEE Transactions on*, 7(1):109–122, Jan 2014.

[14] D. Le-Phuoc, M. Dao-Tran, J. X. Parreira, and M. Hauswirth. A native and adaptive approach for unified processing of linked streams and linked data. In *Proceedings of the 10th international conference on The semantic web - Volume Part I*. Springer-Verlag, 2011.

[15] D. Luckham. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. In *Interchange and Reasoning on the Web Rule Representation*, volume 5321 of *Lecture Notes in Computer Science*, pages 3–3. Springer Berlin / Heidelberg, 2008. $10.1007/978-3-540-88808-6_2$.

[16] L. Mei, W. Chan, and T. Tse. An adaptive service selection approach to service composition. In *Web Services, 2008. ICWS '08. IEEE International Conference on*, pages 70–77, Sept 2008.

[17] R. Parker. *Missing Data Problems in Machine Learning*. VDM Verlag, 2010.

[18] H. Qu, M. Song, R. Wang, W. Qu, X. Luo, P. Zhao, and J. Song. An adaptive service composition performance guarantee mechanism in peer-to-peer network. In *Computer Network and Multimedia Technology, 2009. CNMT 2009. International Symposium on*, pages 1–4, Jan 2009.

[19] L. Wang, L. Rui, X. Qiu, W. Li, and K. Jiang. A self-adaptive recovery strategy for service composition in ubiquitous stub environments. In *Computers and Communications (ISCC), 2013 IEEE Symposium on*, pages 000892–000897, July 2013.

[20] L. Yu, W. Zhili, M. Lingli, W. Jiang, L. Meng, and Q. Xue-song. Adaptive web services composition using q-learning in cloud. In *Ninth IEEE World Congress on Services (SERVICES)*, pages 393–396, June 2013.