Matching Customer Requests to Service Offerings in Real-Time

Marcel Tilly
European Microsoft Innovation Center
Ritterstrasse 23
Aachen, Germany

marcel.tilly@microsoft.com

Stephan Reiff-Marganiec
Department of Computer Science
University of Leicester
Leicester, UK

srm13@le.ac.uk

ABSTRACT

Classic request-response Service-oriented architecture (SOA) has reached a level of maturity where SOA inspired extensions are enabling new and creative domains like the Internet of Things, real-time business or real-time Web. These new domains impose new requirements on SOA, such as a huge data volume, mediation between various data structures and a large number of sources that need to be procured, processed and provided with almost zero latency. Service selection is one of the areas where decisions have to be made based on consumer requests and service offerings. Processing this data requires typical SOA behavior combined with more elaborate approaches to process large amounts of data with near-zero latency. The approach presented in this paper combines pub-sub approaches for processing service offerings and mediations with classical request-response SOA approaches for consumer requests facilitated by Complex Event Processing (CEP). This paper presents a novel approach for subscribing to dynamic service properties and receiving up-to-date information in real-time. Therefore, we are able to select services with near-zero latency since there is no need to pull for property values anymore. The paper shows how to map requests to streaming data, how to process and answer complex requests with low latency and how to enable real-time service selection.

Categories and Subject Descriptors

H3.4 [**Systems and Software**]: Current awareness systems, distributed systems and user profiles and alert services

General Terms

Algorithms

Keywords

Mediation, Service Selection, Complex Event Processing, Non-Functional Properties, SOA.

1. INTRODUCTION

Nowadays businesses as well as the Web require for information to be available in real-time in order to reply to request, make decisions and generally stay competitive. This in turn requires for data to be processed in real-time. In general in service-oriented architecture (SOA) we are less concerned about latency of data processing. Clearly, there are investigations of service-level agreements (SLA) and quality of service (QoS) to guarantee service delivery. Based on this, several approaches on monitoring SLAs have emerged and solutions to find most relevant services for a given context have been developed. Most of this work is assuming that the relevant information for decision making is available and accurate.

Properties for service selection are considered to be non-functional or functional, and the available approaches are based on the fact that properties are pulled from service repositories (that is from service metadata) or possibly from the services directly before the algorithm determines the most relevant service for a given context. Repositories are useful for static data and polling services directly works if a small number of properties of a small number of services is of interest. We believe that there is an emergent need to provide methods to enable the continuous evaluation of functional and non-functional properties especially in the case where the number of services is high [1].

Let's assume there is a user who tries to locate the nearest printer with the shortest print queue because he has a deadline and needs to print out an important report. Therefore, the system needs the location of the user, typically part of a user profile, the geographical location of the printers, and information about the print queue of each printer. In service selection, an algorithm compares the location of the user with the location of the printer taking into account the number of documents in each print queue. There are several approaches which are able to identify the most relevant printer within a given context – so this is not the challenge we are tackling in this paper; we are interested in obtaining the data that is used for the decision making. The geographical location is static information – it does not change continuously over time. We will be using the term *static property* for properties whose values are static over time. The number of documents in the print queue is not static – it is time dependent and changes over time as documents are printed or new documents are added to the queue. Hence the length of the print queue is a *dynamic property*.

It is quite challenging to get an accurate view of this data with classic request-response approaches which are usually employed in SOA. Consider the number of printers within a company, all taxis of a company within a city, or even the shuttle service on a large company campus. Here the number of possible services, namely printers, taxis, or shuttles is high. In addition the length of the print queue or the geo location of taxis or shuttles change very frequently – they are highly dynamic properties. Using a typical request-response approach every time a user asks for a taxi the system has to poll all the taxis' geo locations and other properties just to be able to identify the most relevant one for the request – if we consider that this might be 50 or even 100 taxis we get a feeling for the scale. In such realistic settings it is becoming quite challenging to answer a simple question such as 'find the nearest shuttle to my location' quickly.

We already identified a need for methods for continuously evaluating properties. We can define this more crisply as a need for an approach delivering dynamic service properties at any time to support service selection from huge lists of services.

In this paper we consider the use of complex event processing to enable a real-time view of dynamic service properties to enable a fast and accurate view of their values with an application in real-time service selection. Our approach can be seamlessly integrated with existing service selection approaches. We present a novel architecture, data model and selection process to put the above into practice.

Basically, we propose to combine existing request-response approaches (the pull model, Figure 1(a)) with publish-subscribe techniques (the push model) (Figure 1(b)).

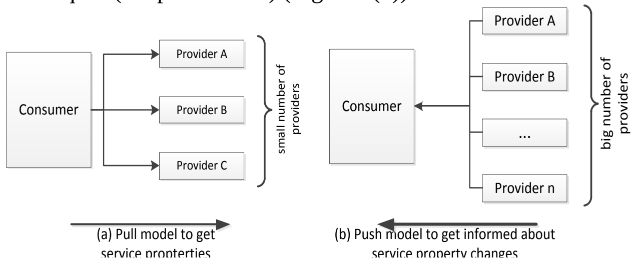
Consumer
Provider A
Provider B
Provider C
small number of providers
Consumer
Provider A
Provider B
...
Provider n
big number of providers
(a) Pull model to get service properties
(b) Push model to get informed about service property changes

Figure 1: Metaphor change from pull to push model

Section 2 presents a motivating example, enforcing the need for the mechanisms presented, while section 3 provides some essential background work. Sections 4 and 5 represent the core of the paper where we introduce the architecture and selection process respectively. Section 6 points to some related work while section 7 concludes the paper and provides an outlook to further work.

2. Motivating Example

We will introduce one example to highlight the necessity of our approach. This approach is useful in all scenarios in which we need to select services from a large set of services which one can usually find in sensor networks, e.g.

- Wireless traffic sensor networks to monitor vehicle traffic on highways or in congested parts of a city.
- Wireless surveillance sensor networks for providing security in shopping malls, parking garages, and other facilities.
- Wireless parking lot sensor networks to determine which spots are occupied and which are free.

In fleet management, like taxi companies, with a large amount of taxis it is almost impossible to use the classical request-response approach to find the nearest taxi for a given user location. Therefore the fleet management must be aware of the taxis location at any given time. But there is usually also no need to store all the provided locations of all taxis forever. The management system only requires the latest data to process a user request to locate the nearest taxi. There is no necessity to persist the data for later use.

In the scenario (see Figure 2) there is a customer with a given geo location requesting a taxi. The fleet management system has to identify the most relevant taxi in terms of (1) availability and (2) proximity to the customer's location. There are two taxis, A and C, which are close to the customer's location but they are not available. Taxi B is the closest which is available. Of course the fleet management could take traffic information into account, and then maybe taxi D becomes the best solution because it is reasonably close, available and might arrive earlier because of beneficial traffic conditions.

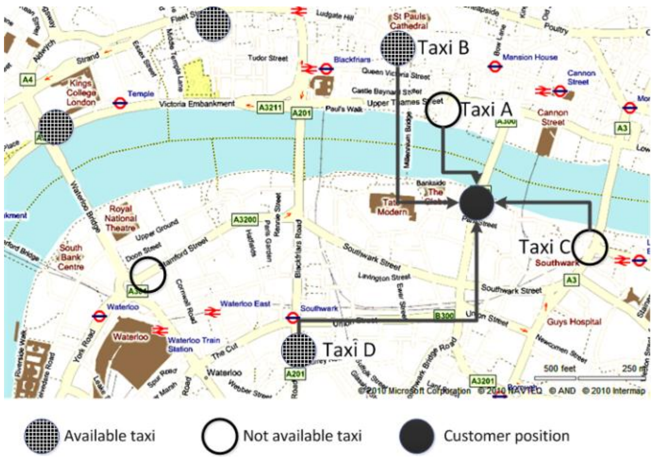

Figure 2: Taxi management using geo location, availability and traffic information

This scenario shows (1) how different kind of properties of taxis (here: availability and geo location) and (2) properties of different services (here: taxi and traffic) are used to select services. Furthermore, the geo location and the traffic information are data which changes rapidly and it does not make sense to store all of this data because it is only relevant when a service has to be selected.

3. Background

This section introduces the basic ideas which we combine to improve service selection and mediation approaches for consumer requests in real-time. Thus, we will also provide a short introduction to complex event processing (CEP) which we use to process dynamic service properties.

Many SOA efforts are focusing on implementing synchronous or asynchronous request-response interaction patterns. This approach works for highly centralized environments and create loose coupling for distributed software components. It tends to create tight coupling and added dependencies for business processes at functional level.

While this is not true on the conceptual level it is still very valid on a technical level. On the conceptual level SOA already achieves loose coupling this is not available on the implementation level. On this level the consumer in most cases is coupled to a concrete service by generating a proxy based on interface definitions, such as WSDL.

Thus, in the migration towards real-time enterprises which are also constantly connected and always available on the web, we have to rethink the current approaches and have to investigate alternative approaches and design patterns in addition to synchronous request-driven SOA.

3.1 Complex Event Processing

Complex event processing (CEP) is the continuous and incremental processing of event streams from multiple sources based on declarative query and pattern specifications in quasi real-time with near-zero latency as described in [2]. CEP is a set of techniques and tools helping to understand and control event-driven information systems. It consists of very simple techniques – a set of old and new ones – from which some are well-known, such as rule-based systems and others are novel techniques, such as tracking causal histories of events in large distributed computer systems. Therefore, the approach of using CEP for our approach is very promising.

A complex event is an event which aggregates incoming source events that are related in various ways, such as by cause, by time, or by membership. CEP makes use of relationships between events to answer questions like:

- “Is our system doing the things it should do?”
- “Will our shipment arrive on time?”
- “Is something going wrong in our production line?”

CEP is applicable to a many information systems and in fact is already used in e.g. analysing click-streams of users in the internet. It helps to define and utilize relationships between events. In addition it is also flexible because a user can specify the events and their relation at any time. In these efforts, the goal is to build a data management system that handles data streams as first class citizens. These systems use SQL like query languages in order to express queries on the data streams. We will be using CEP as a system to process incoming events and provide a real-time view to the subscribed service properties.

4. Basic concepts

In our work, services offer dynamic properties to which consumer can subscribe, such as the dynamic *GeoLocation* property of a taxis service and the number of current passengers from which the system can derive if the taxi is available or not.

We envision that our approach can be adopted easily as it only requires the addition of two interfaces: (1) The publisher endpoint is exposed on the service side to which the consumer can register or subscribe to events and (2) the subscriber endpoint is exposed by the consumer to enable the services to fire events in a fire and forget fashion (see Figure 3).

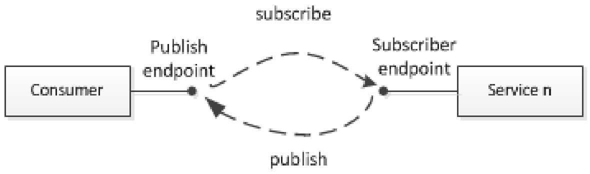

Figure 3: Pub/Sub endpoints

The publisher interface which enables the registry to subscribe to a set of dynamic properties provides two operations:

Subscribe(topic, refresh time, endpoint): Id

- *topic*: the topic to be subscribed to, using dot notation such as *Dynamic.GEOLocation*
- *refreshTime*: how often should events be send out
- *endpoint*: the endpoint of the publish event operation
- *Id*: Unique registration id for the subscription

Unsubscribe(Id)

- *Id:* Unique registration id

The subscriber interface offered by the consumer provides only one operation:

PublishEvent(event)

An event *event* is a tuple of values $event=<s_e, t_s, t_e, p>$, containing service endpoint address s_e, time information t_s and t_e, and payload p. The time information defines the valid start time t_s and end time t_e of the event and the payload is defined by the type of the subscribed topic. For example the *GeoLocation* could be defined as record with *Longitude* and *Latitude, both* of the XML schema type *xs:int*.

As described in [3] processing of streaming data is an important practical problem that arises in time-sensitive applications where the data must be analyzed as soon as they arrive, or where the large volume of incoming data makes storing all data for future analysis impossible. Stream processing has become a hot research topic in several areas including stream data mining, stream database or continuous queries, and sensor networks.

We define static properties p_s as constant over time, such as a location of a printer, the vendor of a printing machine, or the number of a taxi etc. Dynamic properties p_d are changing over time. Using these, we define non-functional properties *NFP* as a tuple of static properties and dynamic properties:

$$NFP(t)=<p_s, p_d>.$$

For the fleet management scenario the schema of the non-functional properties might look as follows:

```
<NFProperties>
  <Static>
    <TaxiId type=”xs:string”/>
  </Static>
  <Dynamic>
```

```
    <GEOLocation>
      <Longitude type=”xs:int”/>
      <Latitude type=”xs:int”/>
    </GEOLocation>
    <PassengerNumber type=”xs:int”/>
  </Dynamic>
</NFProperties>
```

This presents the static data schema; like a snapshot in time. Temporal aspects are covered by events and therefore we would see different data at different point in time.

Since temporal dynamic properties are defined as time dependent we can see them as discrete events and use standard temporal algebra approaches to reason over them. Current temporal algebra research and solutions are focusing on complex event processing. Therefore, we can use on consumer side rule based approaches to select and project events from data streams. SQL-like syntax can be used to express complex aggregations and correlations on those event streams, such as

Select e from s where Op(e)

with
e: event from stream
s: event stream
Op: Operation on events from stream

5. Architecture

As a central instance we still use a Registry. This Registry encapsulates the processing of the incoming request from consumer side and the incoming events from service side and maps both. To setup the system there is a need that for a potential consumer request (here: Find a taxi) the system has to identify all services and subscribe to the relevant non-functional properties which will support our service selection during runtime (see Figure 4).

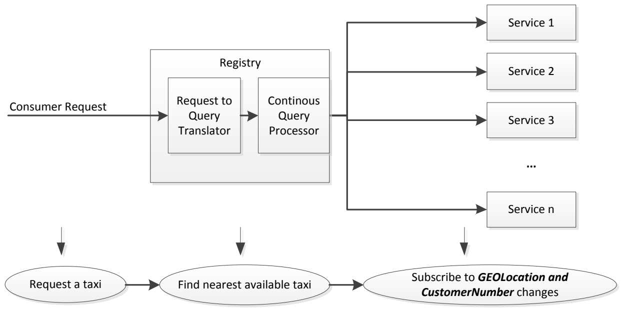
Registry
Consumer Request
Request to Query Translator
Continous Query Processor
Service 1
Service 2
Service 3
...
Service n
Request a taxi
Find nearest available taxi
Subscribe to GEOLocation and CustomerNumber changes

Figure 4: Concept of using pub/sub for service selection

Thus, during runtime the Registry is receiving continuous streams of events from subscribed services. Then, an incoming consumer request is handled as a query on subscribed service properties.

Instead of pulling at request time all the data from all services the registry knows at any time the status of all services. Therefore, this allows for service selection in real-time independent of the number of services.

An event will contain metadata and payload. The metadata contains information about the time when the event was created on publisher side. We are enriching this time information also with subscriber time information when the event enters the subscribers system. The payload is defined by the schema of the subscribed topic, such as *GeoLocation* containing *Longitude* and *Latitude*.

5.1 Request Mediator

The Request Mediator exposes an endpoint to collect all incoming events from registered services. Its responsibility is to normalize the incoming data streams. Usually, not all events provide the same data structure therefore the Request Mediator maintains a mapping table to transform incoming events from endpoints into a normalized data stream. Let’s assume the service1 provides events containing *GEOLocation* and *availability* while service service3 provides the data as *MyLocation* and *Customer_Number*. In our current implementation we are simply using XSLT scripts to normalize event streams internally before the event data is forwarded to the Information Mediator.

5.2 Information Mediator

The Information Mediator maps consumer request to queries on continuous event streams provided by the request mediator. On the consumer side the framework still offers a normal Web Service interface which internally needs to transform into a query which is executed over the event stream. Ideally this queries are not hard coded somewhere but they are stored in a repository to be adaptable during runtime.

The Information Mediator also ensures the quality of the events from event streams, such as duplicated events or out-of-order events. Here, our approach benefits from the CEP work. The specific time information we are adding to the event helps to control the quality of events and result. While valid start time and valid end time are generated at service side the Information Mediator also added internal time information (called: System time) to the events. Within the Information Mediator internal clock increments are used to move time forward decoupled from external sources. Thus, the order of events is guaranteed and the quality of the results can be ensured. Basically, this is a classical CEP topic (see [4]) and the approach is simply benefiting from using CEP technology here. In addition the Information Mediator is able to detect missing events since the refresh time is set within the subscription process. Here it is possible to apply different policy to react on missing events, such as simply ignore missing events, use the latest event until a new event arrive, or raise an exception because the absent of an event is an exceptional case. How to handle missing events depends on the scenario and does not require a general solution.

Basically, the decoupling of information and requests helps to integrate other flexible work into our solution. Thus, it is easy to improve the request mediator with some more sophisticated Semantic Web Service implementation if needed.

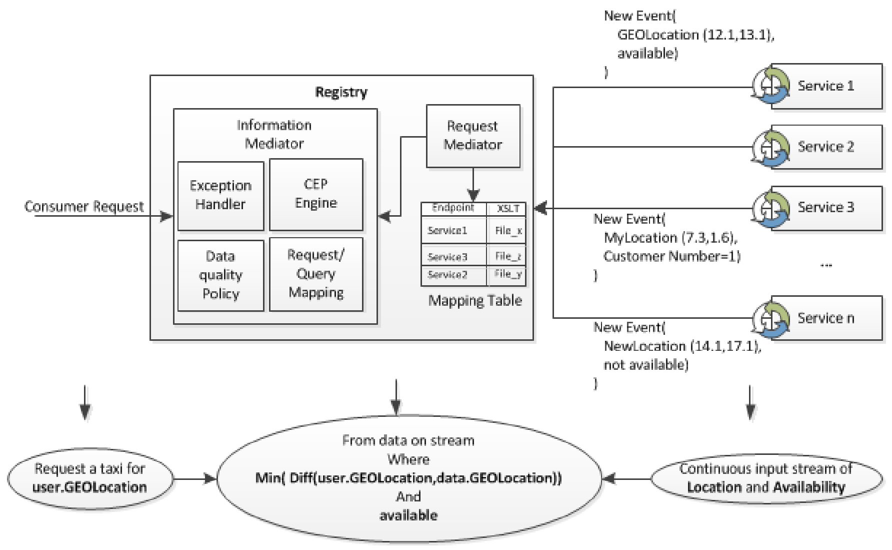
New Event(GEOLocation (12.1,13.1), available))
Registry
Information Mediator
Request Mediator
Exception Handler
CEP Engine
Data quality Policy
Request/ Query Mapping
Endpoint XSLT
Service1 File_x
Service3 File_z
Service2 File_y
Mapping Table
Consumer Request
Service 1
Service 2
Service 3
Service n
New Event(MyLocation (7.3,1.6), Customer Number=1))
New Event(NewLocation (14.1,17.1), not available))
Request a taxi for user.GEOLocation
From data on stream Where Min(Diff(user.GEOLocation,data.GEOLocation)) And available
Continuous input stream of Location and Availability

Figure 5: General architecture of continuous event processing for service selection

6. Validation example

Let us go back to the taxi management scenario to illustrate the presented theory with a simple example. We will not describe how a service (in this case a taxi) is sending messages ('events') via soap request to the request mediator. However, an event looks like this:

```
E₁= <
sₑ='http://www.contoso.com:8080/taxi3';
tₛ=2010-08-20 10:30:30;
tₛ=2010-08-20 10:30:40,
p= <
   MyLocation.Longitude=12;
   MyLocation.Latitude = 10
   Customer_Number = 1
>
>
```

The event E_1 is provided by the endpoint taxi1 with a valid timespan of 10 sec. (t_e-t_s). The payload for taxi3 is *MyLocation* and *Customer_Number*. The Request Mediator tranforms the payload into the normalized payload for the query. Thus, the event E_1' looks like:

```
E₁'= <
sₑ='http://www.contoso.com:8080/taxi3';
tₛ=2010-08-20 10:30:30;
tₛ=2010-08-20 10:30:40,
p= <
   GEOLocation.Longitude=12;
   GEOLocation.Latitude = 10
   Available = false
>
>
```

The Information Mediator adds system time information to the events and checks if other events can be discarded already.

For a user request at a given time t we only have to take into account the events in which time t is part of events' valid time interval (see Figure 6). In the sample this is event e13 for taxi 1, event e22 for taxi 2, and event e32 for taxi 3. The Information Mediator uses only these events to execute the query to find the nearest taxis for the given user location which is available. That's it – one simple query and a reply in real-time.

7. Related Work

While much focus has been given to efficient data processing methods that support complex data needs (expressed for example by queries or user profiles), less attention has been given to efficient data gathering methods in SOA for service selection and mediation.

As already mentioned there is a lot of work about service selection based on non-functional properties. [5] provides a survey and classification of service selection based on non-functional properties. Most of the related work on using non-functional properties for service selection concentrates on defining QoS (Quality of Service) ontology languages and vocabularies and identification of various QoS metrics and their measurements with respect to semantic services. In [6] QoS ontology models are defined while [7] separates different non-functional criteria into different service categories. This is more sensible than ranking all kinds of services by using the same

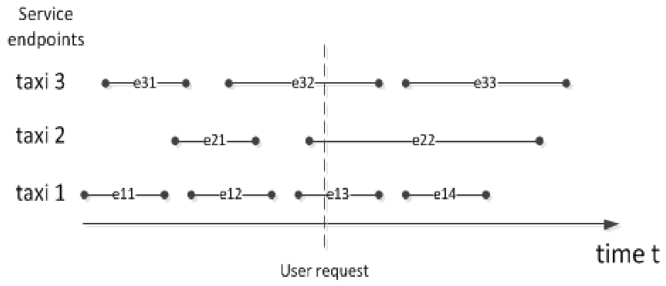

Figure 6: Event streams of taxi 1 to 3

predefined criteria and hence not considering the different attributes that occur with specific services. The key feature of this approach is about incorporating the Logic Scoring of Preferences (LSP) for ranking different services. In [8] there is also a strong focus on efficiency of the algorithm but not on gathering, collecting and aggregating properties for the algorithm itself.

Bonifati et. al. [9] describes a very interesting approach for using active rules for pushing reactive services. The combination of this approach with our approach would need some further investigation but looks promising as an end-to-end solution for pre-filtering on service side to reduce network traffic and to correlate and aggregate on consumer side for real-time service selection and adaptability. Roitman et. al. [10] presents a framework for satisfaction of complex data needs involving volatile data. But the focus is on pull-based environments.

With push based systems, data is pushed to the system and the research focus is mainly on aspects of efficient data processing, where load shedding techniques [11] can be applied in order to control what portions of the pushed data to process and to increase latency. Such systems include publish-subscribe (pub/sub) ([12]), stream processing ([13]), and complex event processing.

But all these systems do not combine their approach with SOA to improve service selection. Pub-sub systems allow the registration of complex requirements at servers and focus mainly on the trade-off between data processing efficiency and the expressiveness of the queries that can be processed by the system. Stream processing systems are also push-based in nature and focus mainly on smart filtering and load shedding techniques. Complex event processing systems assume the pushing of a stream of raw events and focus mainly on efficient complex events and situations identification only.

8. Conclusions and Future Work

We presented a new approach which combines CEP with service selection approaches to enable a new set of scenarios for service selection. Our approach investigates service selection problems with a huge number of potential services and highly dynamic service properties. We presented an easy way to seamlessly integrate our approach into existing service selection. We presented a way to subscribe to specific dynamic properties. This enables the service selection to be faster in their selection process and more accurate by having more real-time data. This was achieved by replacing the classical pull approach with a push approach.

The next steps are to provide more validation results and to extend it towards situations where services do not offer exactly the same dynamic properties. We believe that we can easily adapt work from semantic web and mediation approaches. We will also investigate into the usage of formal temporal algebra to ground dynamic NFP-based selections on a valid formal model.

In addition we will also investigate if the approach can also improve service composition by using context information or user data in real-time.

9. References

[1] D. Chou, "Using Events in Highly Distributed Architectures," *The Architecture Journal*, 2008.

[2] D. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*, Amsterdam: Addison-Wesley Longman, 2002.

[3] A. Riabov and Z. Liu, "Scalable planning for distributed stream processing systems," *Proceedings of ICAPS*, 2006.

[4] R.S. Barga, J. Goldstein, M. Ali, and M. Hong, "Consistent Streaming Through Time : A Vision for Event Stream Processing 2 . CEDR Temporal Stream Model," *General Systems*, 2007.

[5] H. Yu and S. Reiff-Marganiec, "Non-functional property based service selection: A survey and classification of approaches," *Proc. of 2nd Non Functional Properties and Service Level Agreements in SOC Workshop (NFPSLASOC'08)*, Citeseer, 2008.

[6] I. Papaioannou, D. Tsesmetzis, I. Roussaki, and M. Anagnostou, "A QoS ontology language for web-services," *20th International Conference on Advanced Information Networking and Applications, 2006. AINA 2006*, 2006, p. 6 pp.

[7] S. Reiff-Marganiec, H. Yu, and M. Tilly, "Service selection based on non-functional properties," *Service-Oriented Computing-ICSOC 2007 Workshops*, Springer, 2009, p. 128–138.

[8] T. Yu, Y. Zhang, and K. Lin, "Efficient algorithms for Web services selection with end-to-end QoS constraints," *ACM Transactions on the Web*, vol. 1, 2007, pp. 6-es.

[9] A. Bonifati, S. Ceri, and S. Paraboschi, "Pushing reactive services to XML repositories using active rules," *Computer Networks*, vol. 39, 2002, pp. 645-660.

[10] H. Roitman, A. Gal, and L. Raschid, "Web Monitoring 2.0: Crossing Streams to Satisfy Complex Data Needs," *Proceedings of the 2009 IEEE International Conference on Data Engineering*, IEEE Computer Society, 2009, p. 1215–1218.

[11] Y.T. Song, L. Sunil, P. Bin, and W. Lafayette, "Load Shedding in Stream Databases : A Control-Based Approach," *Framework*, pp. 787-798.

[12] A. Demers, J. Gehrke, M. Hong, M. Riedewald, and W. White, "Towards expressive publish/subscribe systems," *Advances in Database Technology-EDBT 2006*, 2006, p. 627–644.

[13] D. Abadi, D. Carney, U. Cetintemel, and M, "Aurora: a data stream management system," *Management of data*, 2003.