PLAY: Semantics-Based Event Marketplace

Roland Stühmer, Yiannis Verginadis, Iyad Alshabani, Thomas Morsellino, Antonio Aversa

PLAY: Semantics-Based Event Marketplace

Roland Stühmer[1], Yiannis Verginadis[2], Iyad Alshabani[3], Thomas Morsellino[4], Antonio Aversa[5]

[1]FZI Forschungszentrum Informatik, Karlsruhe, Germany
[2]ICCS, National Technical University of Athens, Greece
[3]INRIA-I3S-CNRS, University of Nice-Sophia Antipolis, France
[4]LINAGORA, Toulouse, France
[5]Orange Labs SOP/BIZZ/DIAM, Sophia-Antipolis, France

Abstract. In this paper we present PLAY Platform, a Web-oriented distributed semantic middleware that serves as an Event Marketplace: the place where heterogeneous events can be integrated and combined. The purpose of the platform is to derive useful information from diverse real-time sources such as collaborative processes. The platform provides technology where instant results are needed or where heterogeneous data must be integrated on the fly or where the data arrive fast enough to require the stream processing nature of our approach. The main advantages of the platforms are its scalability (cloud-based nature) and the expressivity of the event combinations that can be defined (using both real-time and historical data). The platform has been applied in a use case about Personal data management. In this paper we present some results from the validation, focusing on smartphone and social media integration.

Keywords: Mobile Data, Linked Data, Personal Data, Real-time Web, Event Marketplace, Complex Event Processing, Semantic Streams

1 Introduction

Recently, there has been a significant paradigm shift towards real-time computing. Previously, requests for Web sites just like queries against databases were concerned with looking at what happened in the past. On the other hand, complex event processing (CEP) is a technology concerned with processing real-time events, i.e., CEP is concerned with what has just happened. An event is something that has happened, or is contemplated as having happened [8]. For example, an event may signify a sensor reading, and so forth. Using complex event processing this paper wants to outline a framework for dynamic and complex, event-driven interaction for the Web. This infrastructure leads to the concept of the **Event Marketplace** (similar to a service marketplace) where events coming from different event producers (as illustrated above) can be arbitrary combined by different event consumers. The most important requirement is to build the Marketplace upon widely-accepted open standards that enable different components of the event processing architecture to be plugged into an event-processing "fabric" with minimal effort, allowing the development of time-driven, event-based, global applications. This "on-the-fly-adaptive" nature of the Marketplace will enable the dynamic definition of situations of interest (complex

adfa, p. 687, 2011.

event patterns) and ad-hoc generation of timely reactions to new situations. In this paper we present the concept and the architecture of a platform/middleware that can satisfy some of these requirements today. We argue that the proposed solution can scale regarding the distribution of services (sources) and the throughput of interesting information that can be exchanged and can be easily extended with new services (openness).

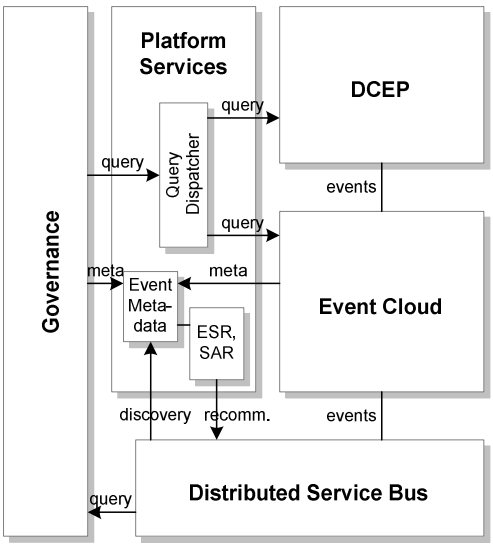

Fig. 1: Conceptual Architecture

2 Platform

The conceptual architecture for our platform is depicted in Fig. 1. We introduce the components briefly. The **Distributed Service Bus** (DSB) at bottom right of the figure provides the SOA and EDA (Event Driven Architecture) infrastructure for components and end user services. The **Governance** component allows users to get information about services and events. The **Event Cloud** provides storage and forwarding of events. Its role is manipulating events, real-time or historic. Real-time subscriptions may use a simple set of operators such as conjunctive queries to filter out an interesting event. More complex queries are executed in the DCEP component. The **DCEP** component (Distributed Complex Event Processing) has the role of detecting complex events by means of event patterns. To detect complex events, DCEP gets simple events from then Event Cloud as defined in the event patterns. The pattern language is BDPL which we introduce below. The **Platform Services** incorporate several additions to the platform. The Query Dispatcher has the role of decomposing and deploying patterns in the Event Cloud and DCEP. The Event Metadata component enables the discovery of relevant events for a consumer and provides data to the subscription recommender. The recommenders Event Subscription Recommender (ESR) and Service Adaptation Recommender (SAR) are discussed below.

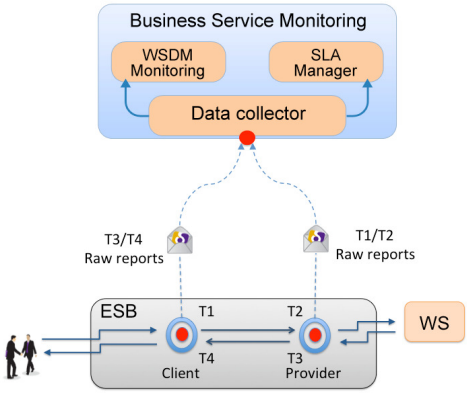

Fig. 2: Raw Reports time stamps illustration

2.1 Governance and Monitoring

The Business Service Monitoring Architecture is an implementation that is called EasierBSM[1]. It enables the monitoring of the Distributed Service Bus (DSB) middleware: services that are deployed on bus as well as external applications. The mechanism is event-driven, monitoring exclusively receives data from the bus as notifications. Components generate and send 'reports' about their activity.

Raw Reports Events. Raw reports (Fig. 2) are sent by interceptors (located around services of the business node). They are based on a model defined by Linagora[2]. T1/T2 report is sent, giving the status of the exchange (i) before the client request is sent to the provider and (ii) after the provider receives the client request. A T3/T4 report is sent, giving the status of the exchange (iii) before the provider sends the response to the client and (iv) after the client receives the response. Several experiments have led to an optimal configuration of 2 events per exchange. Actually, reports are sent on T2 and (potentially if the exchange pattern is In-Out) T4. This choice minimizes the number of non-functional events while sustaining an efficient monitoring. Indeed, it would be possible to send only one Raw Report containing the 4 time stamps, but in this case, the EasierBSM would not be able to manage non-responding services.

Service Level Agreement Events. The Service Level Monitoring is dedicated to the technical aspects and is the lowest kind of event EasierBSM is able to send. For that purpose, EasierBSM contains two components (see Fig. 2) implementing respectively the WSDM [20] (WSDM Monitoring component) and WS-Agreement[3] (SLA Manager component). Both of them use the Raw Reports events as data input. Once QoS are computed, it is possible to build and negotiate agreements at a governance level between service consumer and provider. Negotiated Service Level Agreements are loaded in the SLA component. It receives also the Raw Reports from the DataCollector and checks if a particular exchange is violating an agreement. Then, an SLA alert is potentially sent as an upper level monitoring notification using the Common Alerting Protocol (CAP[4]) standard.

2.2 Event Cloud

The EventCloud [21] is a distributed datastore that allows to store quadruples (RDF triples with context) and to manage events represented as quadruples or set of quadruples (a.k.a., event). To scale, the architecture is based on a structured Peer-to-Peer (P2P) network named Content Addressable Network (CAN) [22]. A CAN is a structured P2P network (structured in opposition to unstructured, another category of P2P networks better suited to high peer churn) based on a d-dimensional Cartesian coordinate space labeled *D*. This space is dynamically partitioned among all peers in the system such that each node is responsible for indexing and storing data in a zone of *D*

[1] http://research.linagora.com/display/easierbsm
[2] http://research.linagora.com/display/esstar/Es-RawReport
[3] http://schemas.ggf.org/graap/2007/03/ws-agreement
[4] http://docs.oasis-open.org/emergency/cap/v1.2/

thanks to a traditional RDF datastore such as Jena. According to our data model, we use a 4-dimensional CAN in order to associate each RDF term of a quadruple to a dimension of the CAN network. Thus, a quadruple to index is a point in a 4-dimensional space.

Retrieval Model At the EventCloud level, an API is provided according to a retrieval model based on *pull* and *push* mechanisms. The *pull* or *put/get* mode refers to one-time queries; an application formulates a query to retrieve data which have been already stored. In contrast, the *push* or *pub/sub* mode is used to notify applications which register long standing queries and push back a notification each time an event that matches them occurs. Both retrieval modes have their filter model based on SPARQL. We allow the SELECT query form and a pattern applies to one graph value at a time. As such SPARQL provides us the ability to formulate a subscription by associating several filter constraints to a quadruple, but also to a set of quadruples that belong to the same event. This means that several quadruples of an event that are published asynchronously at different times may participate in the matching of a subscription by using their common constraints. Also, due to the distributed nature of the EventCloud, each quadruple is possibly stored in different peers. It is important to understand that our push retrieval mode is not supposed to act as a CEP engine correlating several events from several streams.

2.3 Big Data Processing Language

To combine real-time data and contextual (historic) data we propose a language called Big Data Processing Language (BDPL). The language is suited to be deployed in a distributed setting. We modelled BDPL as close to SPARQL 1.1 as possible. Bound variables in the query are allowed to join events based on values from the individual event instances. From the SPARQL 1.1 language we use the following subset: CONSTRUCT queries without operators UNION and subqueries[5], OPTIONAL or LIMIT[6] clauses. We distinguish graph patterns (using syntax EVENT and GRAPH) into real-time data and historic data respectively. The real-time parts may be combined with temporal operators such as time windows from [10] to enable temporal processing. Intuitively, a query is fulfilled if there is a mapping [13] for all variables from the real-time parts and a *compatible* mapping for the historic data at the time of the real-time answer. Operationally this means that the real-time part is applied to the streams. When there is a result the variable mapping will be checked for compatibility with all historic parts in the query.

The model for the query language also requires a model for the data to be queried i.e., the events. The Linked Data principles [14] are a methodology of publishing structured data and to interlink the data to make them more useful. These principles apply to event modelling. We use an event schema from which the event types are inherited. The schema [15] makes use of related work by reusing the class "Event" from Dolce Ultralight based on DOLCE [4]. We are developing this event model to

[5] These operators can be emulated using two or more separate queries.
[6] These operators do not make sense on streams i.e., potentially unbounded datasets.

satisfy requirements of an open platform addressing *variety* in big data. As such, data from the Web must be reused and be extensible for broader participation.

2.4 ESR and SAR Recommenders

In this section, we focus on ESR and SAR Recommenders that are included in the platform services conceptual component (Fig. 1). We aimed to support dynamic recommendations of new event streams to which a service should subscribe for a meaningful period of time, in order to take advantage of situational information, expressed as simple or complex events. We developed ESR that exploits real-time event streams in order to dynamically produce new event subscriptions. This component provides added value assistance to services and users that will have the choice to be subscribed to the "right" event streams at the "right time" and for the "proper period of time". Thus, ESR improves subscription efficiency and prevents from unnecessary network traffic and additional workload on the subscriber's side. SAR is the second recommender. It addresses the issue of business process adaptation. For SAR we enhanced aspect-oriented business process management [17] with event-driven capabilities for discovering situations requiring adaptations. To this end, we developed an aspect-oriented extension to BPMN2.0 similar to AO4BPMN [18]. In order to cope with the advanced event-driven and situational processing related needs of both these recommenders, we developed a goal-driven, ECA-based hierarchical framework, called Situation-Action-Network (SAN) as background mechanism. It is a modeling framework that can be used for defining systems' reactions to significant situations with the purpose of fulfilling or satisfying a goal. Recently, the SAN modeling framework has been extended in order to alleviate a part of the modeler's effort during the design of SAN trees and increase their run-time flexibility [23].

3 Missed Calls Manager Use Case

The *Missed Calls Manager* (shortened hereafter to MCM) shows the event handling as a service in the Android Platform, sensing local telco and social events and acting as a joining link between users and the PLAY Platform that will "mix" events of different natures.

3.1 Use Case scenario and design

"After 3 out/ingoing missed calls to/from the same callee/er in the last 3 minutes, send a recommendation suggesting the user to contact his friend in a different way".

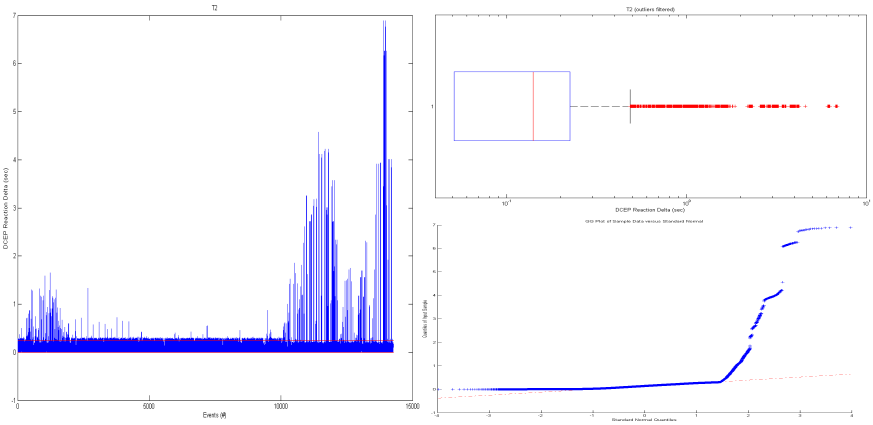

Fig. 3: Normal, box and QQ plots for the KPI T2

In this scenario, for each detected missed call, the MCM App sends to the PLAY DSB a simple event. Each event is delivered through the EventCloud to DCEP, which processes all of them for checking if a sequence of 3 call events with equal caller, callee and direction is occurring. The body of the event processing implementation in BDPL looks as follows, cf. Listing 1. Every time the pattern is detected, the `CONSTRUCT` defines a new complex event: `Recommendation`, pushed back to the Android Device, containing the wished suggestion. This closes the "event loop" for the user, who can decide whether to follow such a suggestion or not.

3.2 Evaluation and performance

In order to evaluate the DCEP performance, the following KPI has been defined and measured in a scenario simulating 41 users randomly generating events and triggering the described pattern. The KPI characterizes the speed of DCEP as the delta between when the last simple event entering the DCEP and when the DCEP build and emits a complex event. The diagrams in Fig. 3 show this KPI with linear normal plot, the logarithmic box plot and the QQ plot, as a function of the cardinality of the events sent. The linear plot shows that the processing speed has initial spikes. These are due to some lazy pro-

```
CONSTRUCT {
    :e rdf:type :Recommendation .
    :e :stream <http://...TaxiUCESRRecomDcep#stream>.
    :e :message "Missed 3 calls, try to contact the
        callee in another way"^^xsd:string .
    :e :members ?e1, ?e2, ?e3 .
    :e uctelco:callerPhoneNumber ?alice .
    :e uctelco:calleePhoneNumber ?bob .
    :e uctelco:answerRequired "true"^^xsd:boolean .
    :e uctelco:action <blank://action1> .
    <blank://action1> rdf:type uctelco:OpenTwitter .
    <blank://action1> :screenName ?screenName .
} WHERE {
    WINDOW {
        EVENT ?id1 {
            ?e1 rdf:type :UcTelcoCall .
            ?e1 :stream <http://...TaxiUCCall#stream> .
            ?e1 uctelco:callerPhoneNumber ?alice .
            ?e1 uctelco:calleePhoneNumber ?bob .
            ?e1 uctelco:direction ?direction .
            ?e1 :screenName ?screenName .
        } SEQ EVENT ?id2 {
            ?e2 rdf:type :UcTelcoCall .
            ?e2 :stream <http://...TaxiUCCall#stream> .
            ?e2 uctelco:callerPhoneNumber ?alice .
            ?e2 uctelco:calleePhoneNumber ?bob .
            ?e2 uctelco:direction ?direction .
        } SEQ EVENT ?id3 {
            ?e3 rdf:type :UcTelcoCall .
            ?e3 :stream <http://...TaxiUCCall#stream> .
            ?e3 uctelco:callerPhoneNumber ?alice .
            ?e3 uctelco:calleePhoneNumber ?bob .
            ?e3 uctelco:direction ?direction .
        }
    } ("PT1M"^^xsd:duration, sliding)
}
```

Listing 1: BDPL Event Pattern

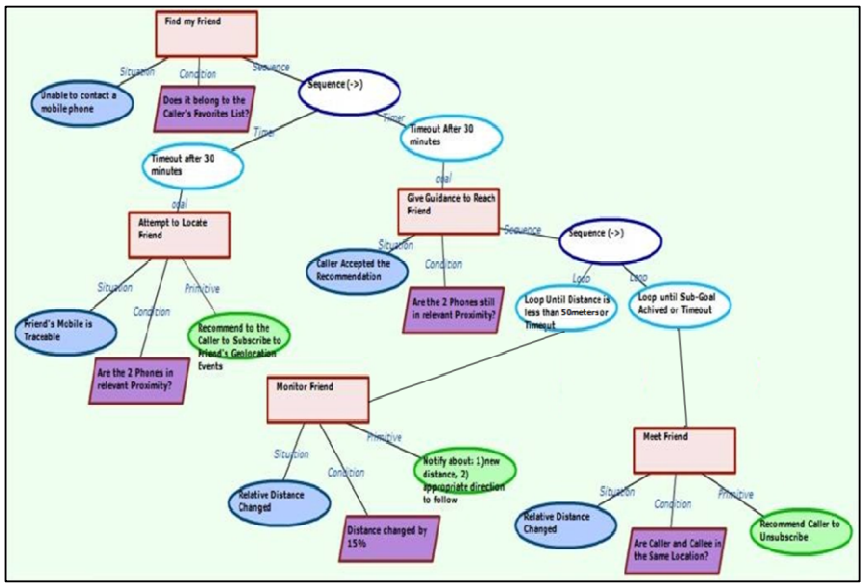

Fig. 4: "Find my Friend" SAN

cessing e.g., DCEP is only connecting to its output streams after a respective event was detected for the first time. The spikes to the right of the plot are due to current event input slowdowns which influence DCEP performance. We are working to correct these in our middleware.

3.3 ESR in Smartphone Scenario

This scenario also involves events transmitted to our platform from the MCM App. This time ESR was introduced for recommending to a user that is unable to contact her friend (e.g. she is in a crowded place, doesn't answer her phone while they have scheduled to meet), the most efficient way to physically meet her. This involves: i) the subscription at the appropriate time to his friend's geolocation events; ii) the calculation of the appropriate duration of this subscription; and iii) the transmission of notifications that can guide the user.

In Fig. 4, the SAN is presented as designed using a dedicated SAN editor. The SAN engine starts the tree traversal when the situation "Unable to contact a mobile phone" is detected (i.e. 3 missed calls events in the last 3 minutes) by our platform. Since the two users are in vicinity (i.e. less than 5 km), ESR recommends the subscription to friend's geolocation events (Fig. 5). The unsubscription takes place once the two users meet.

4 Related Work

Some attempts were made to define a universal vocabulary for events which extends structs of primitive types. One notable approach is the XML format of AMIT presented in [2] providing more detailed temporal semantics and modelling not only events but their generalization, specialization and other relationships between events which can be used in processing. We support such relationships in our system. While designing an event-based system at Internet-scale, we employed widely available Semantic Web technologies to model events as proposed by Sen et al. [3]. There are models for events such as E* [7], F [5] and LODE [6], all of which rely on the DOLCE [4] top-level ontology as we do. However, they do not seem to be tailored to real-time processing because a lot of (e.g. temporal) expressivity such as relative and vague time is not supported by the state of the art in real-time processing engines.

C-SPARQL is a language and a system to process streaming RDF data incrementally [11]. RDF triples are events. Timestamps are attached to the triples implicitly when the events enter the system. Sets of events are matched in windows. This means that the approach has a *set-at-a-time* semantics. EP-SPARQL [9] is built on top of the Prolog-based event processing engine ETALIS [10] like our work. EP-SPARQL supports more event processing operators than C-SPARQL including *event-at-a-time* operators like the sequence of two events which require no mandatory window definition and are thus more declarative. Like C-SPARQL, however, this approach considers events as triples not as objects with further structure. This means that e.g. time is a second-class citizen and not part of the event to be transmitted across distributed systems. BDPL works with structured events consisting of many RDF triples per event as opposed to one triple per event. In addition to real-time data both C-SPARQL and EP-SPARQL can combine stream results with background knowledge. However, they do not propose a distributed system like EventCloud to address the *volume* of data.

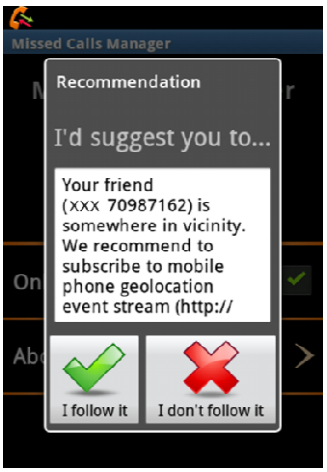

Fig. 5: ESR Recommendation on User's Smartphone

5 Conclusion

We presented a novel approach for real-time querying enabling the expression of complex situations to be notified to the user instantaneously. Main novelty in the approach is the combination of real-time and distributed historical data in the system. The query engine is realized in a distributed manner using event processing and semantic technologies. This work is part of the development of an Event Marketplace, a scalable infrastructure for exchanging and processing heterogeneous events. We argue

that approaches like this will change searching on the Web in real-time, opening possibilities for new scenarios such as acting ahead of time leading to proactivity.

References

1. Fidler, E.; Jacobsen, H.-A.; Li, G. & Mankovski, S. (2005), The padres distributed publish/subscribe system, *in* 'In Feature Interactions in Telecommunications…', pp. 12--30.
2. Adi, A.; Botzer, D. & Etzion, O. (2000), Semantic Event Model, *in* 'ECIS'.
3. Sen, S. & Stojanovic, N. (2010), GRUVe: A Methodology for Complex Event Pattern Life Cycle Management, *in* 'Advanced Information Systems Engineering', Springer, pp. 209-223.
4. Gangemi, A.; Guarino, N.; Masolo, C.; Oltramari, A. & Schneider, L. (2002), Sweetening Ontologies with DOLCE, *in* 'EKAW 2002', Springer-Verlag, London, UK, pp. 166--181.
5. Scherp, A.; Franz, T.; Saathoff, C. & Staab, S. (2009), F–a model of events, *in* ACM
6. Shaw, R.; Troncy, R. & Hardman, L. (2009), LODE: Linking Open Descriptions of Events in 'The Semantic Web', Springer Berlin / Heidelberg, pp. 153-167.
7. Gupta, A. & Jain, R. (2011), Managing Event Information, Morgan & Claypool Publishers.
8. Etzion & Niblett: Event Processing in Action. Manning Publications Co., 8 (2010).
9. Anicic, D.; Fodor, P.; Rudolph, S. & Stojanovic, N. (2011), EP-SPARQL: A Unified Language for Event Processing and Stream Reasoning, *in* 'WWW 2011'.
10. Anicic, D.; Fodor, P.; Rudolph, S.; Stühmer, R.; Stojanovic, N. & Studer, R. (2011), ETALIS: Rule-Based Reasoning in Event Processing, Springer, pp. 99-124
11. Barbieri, D. F et al. (2010), 'Querying RDF streams with C-SPARQL', SIGMOD Rec. 39(1)
12. Shinavier, J. (2010), 'Optimizing real-time RDF data streams', ArXiv e-prints.
13. Pérez, J.; Arenas, M. & Gutierrez, C.(2009), 'Semantics and complexity of SPARQL', ACM.
14. Berners-Lee, T. (2006), 'Linked Data', http://www.w3.org/DesignIssues/LinkedData.html.
15. Harth, A. & Stühmer, R. (2011), 'Publishing Event Streams as Linked Data', Technical report, Karlsruhe Institute of Technology, http://km.aifb.kit.edu/sites/lodstream/.
16. Patiniotiakis, I., Papageorgiou, N., Verginadis, Y., Apostolou, D., Mentzas, G.: Dynamic Event Subscriptions in Distributed Event Based Architectures. ESWA, Elsevier, (2013)
17. Charfi, A., Mezini, M.: Aspect-Oriented Workflow Languages. In: Meersman, R., Tari, Z. (eds.) OTM 2006. LNCS, vol. 4275, pp. 183–200. Springer, Heidelberg (2006)
18. Charfi, A., Mezini, M.: AO4BPEL: An Aspect-Oriented Extension to BPEL. World Wide Web Journal: Recent Advances on Web Services, special issue (2007)
19. Lesbegueries J.; Ben Hamida A.; Salatge N.; Zribi S. & Lorré J-P. (2012) Multilevel event-based monitoring framework for the petals bus: industry article, *in* DEBS 2012, 48-57.
20. Bullard V. & Vambenepe W. (2006) WSDM-MUWS v1.1
21. L. Pellegrino, F. Baude, and I. Alshabani. Towards a Scalable Cloud-based RDF Storage Offering a Pub/Sub Query Service CLOUD COMPUTING 2012, page 243--246. (2012)
22. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. ACM SIGCOMM 31(4), 161–172 (2001)

23. Patiniotakis, I., Papageorgiou, N., Apostolou, D., Verginadis, Y., Mentzas, G.: Collaborative Process flexibility using Multi-Criteria Decision Making. PRO-VE'13, Dresden (2013)