View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by INRIA a CCSD electronic archive server

# Efficient Data-Intensive Event-Driven Interaction in SOA

Quirino Zagarese, Gerardo Canfora, Eugenio Zimeo, Iyad Alshabani, Laurent Pellegrino, Françoise Baude

**To cite this version:**

Quirino Zagarese, Gerardo Canfora, Eugenio Zimeo, Iyad Alshabani, Laurent Pellegrino, et al.. Efficient Data-Intensive Event-Driven Interaction in SOA. SAC '13, the 28th Annual ACM Symposium on Applied Computing, Mar 2013, Coimbra, Portugal. pp.1907-1912, 10.1145/2480362.2480715 . hal-00918637

**HAL Id: hal-00918637**

**https://hal.inria.fr/hal-00918637**

Submitted on 17 Dec 2013

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Efficient Data-Intensive Event-Driven Interaction in SOA

Quirino Zagarese,
Gerardo Canfora,
Eugenio Zimeo
Department of Engineering
University of Sannio
Benevento, Italy
first.last@unisannio.it

Iyad Alshabani,
Laurent Pellegrino,
Françoise Baude
INRIA, I3S-CNRS Université
de Nice Sophia Antipolis,
France
first.last@inria.fr

## ABSTRACT

This paper presents a middleware that enables the efficient delivery of events carrying large attachments. We transparently decouple event-description from event-data, in order to avoid useless data-transfers and modifications to endpoints business logic. Our solution relieves the event-delivery system of large data transfers, by enabling direct, but transparent, publisher to subscriber data-exchange. The experiments show that we can reduce the average event delivery time by half, compared to a standard approach requiring the full mediation of the event-delivery system.

## Keywords

web-services, SOA, EDA, publish-subscribe, data-intensive applications

## 1. INTRODUCTION

Service Oriented Architecture (SOA) has represented an important milestone in software architecture evolution in supporting flexible design of complex and multi-organization applications [4]. Loose coupling and interoperability among software components constitute the main drivers of this architectural model: the former is achieved by spatially decoupling services through specific mediators, such as registries and brokers, the latter by exploiting standard protocols and semantics.

In spite of its flexibility, the architectural model is typically implemented by exploiting procedural programming models, which emphasize remote service calls and workflows. Even though this model can be pragmatically applied to a large class of applications, it fails in some domains where events represent first-class concepts to deal with.

Many computer systems, especially embedded ones, are designed to respond to events: the thermostat signals a low value of the environmental temperature and sends a command to turn on the boiler. However, up to now, many of the systems whose logic is based on external events have been implemented in limited areas and often they are invisible to the user. As computer systems become more interconnected they start to handle an increasing number of events (e.g. an order management system may receive orders from a web site and notify other systems, such as the financial one, to check for example whether a credit card is valid, and the warehouse, to verify that inventory to fulfill the order is present). In this new scenario, some new properties characterize software systems: event propagation (events are propagated to any interested party that is listening to some events to process); timeliness (systems publish events as they occur instead of storing them locally); asynchrony (the system that fires an event does not wait for the receiving system). These properties significantly change the behavior of SOA-based systems. Call stack based interaction assumes that one thing happens after another, identifying a single path of execution where the caller's does not continue to run until the called method completes. If invocations are slow or carry a large amount of data, call stack based interactions become inefficient, and if the services to call are not known a priori, even poorly flexible. Communicating through events, on the other hand, introduces an important shift of responsibility. It allows components (a) to be decoupled, since the caller is no longer aware of the sequence of functions to execute and the components that will execute them, (b) to keep the state which are interested in, since they do not query other systems for information but instead exploit their own copy of the required data.

Events represent state transitions and are commonly modelled as messages comprising a header, that contains message-specific information like priority or expiration time, and a payload, that contains user-specific information [6].

In SOA, event-driven messaging [5] enables the exchange of messages driven by events and subscriptions, thus avoiding inefficient polling interactions. Typically, these messages can be used to carry documents (Document Message pattern) or files (File Transfer pattern) when state transitions occur [9]; for these reasons, messages typically include attachments to carry large amount of data, as proposed by modern Enterprise Service Bus (ESB) like JBossESB [1].

On the other hand, the bigger is the size of such attachments, the more effort is delegated to the event-delivery service, especially when the amount of events grows. Moreover, the event subscriber may not be able to handle the attachment, thus wasting the resources employed to transfer it.

The main contribution of this paper is an architectural solution supporting transparent publisher-to-subscriber di-

[1]http://www.jboss.org/jbossesb

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
*SAC'13* March 18-22, 2013, Coimbra, Portugal.
Copyright 2013 ACM 978-1-4503-1656-9/13/03 ...$10.00.

rect data transfers. We provide a mean to move the resources belonging to an event from the publisher directly to the subscriber, thus avoiding large data-transfers from/to the event-delivery system. The resources are moved only if needed by the subscriber, in order to avoid useless transfers. The subscriber is not aware of the "lazy" nature of the transfer and accessing the attachments does not imply any further coding effort. The proposed solution extends a pub-/sub infrastructure which allows for semantic description of events, by means of the Resource Description Framework (RDF) data meta-model [11].

The remainder of the paper is organized as follows. Section 2 describes the context that originated this work. Section 3 details our architecture and presents a pub/sub scenario. Section 4 focuses on data-transfers and explains how we avoid the useless ones to increase system efficiency. Section 5 analyses the performances in terms of average event delivery time and shows the improvements that can be achieved by employing our data-transfer technique. Section 6 discusses some meaningful related work. Finally, section 7 concludes the paper and introduces future work.

## 2. MOTIVATING CONTEXT

This work originates from the context of the PLAY [2] project. PLAY is a platform that allows for "event-driven interaction in large highly distributed and heterogeneous service systems". The core of the platform is the EventCloud (EC): a component that offers the possibility, for services, to communicate in a loosely coupled fashion thanks to the pub-/sub paradigm. Subscribers register their interest in some type of events in order to asynchronously receive the ones that are matching their concerns. Events descriptions, inside the EC, are represented as sets of quadruples. Quadruples are in the form of (context, subject, predicate, object) where each element is a dubbed RDF term in the RDF [11] terminology. The context value identifies the data source; the subject of a quadruple denotes the resource, the statement is about; the predicate defines a property or a characteristic of the subject; finally, the object presents the value of the property.

Since the EC supports content-based subscriptions, formulated as SPARQL [15] queries, subscribers can specify fine-grained constraints on each RDF term of quadruples. Each published event is stored on the EventCloud to be retrieved later, by means of a standard RDF datastore. Storing events can be useful to create a knowledge-base that may be used at any time to provide statistics or to correlate events (e.g. by employing a Complex Event Processing engine).

The high level of expressiveness offered by the EC event-format makes it ideal for open environments. On the other hand, the EC has not been designed for the delivery of attachments. In order to keep such expressiveness and to enable attachments delivery, we have extended its features by applying the *Decorator* pattern [17].

The resulting architecture is depicted in Figure 1. Publishers and subscribers do not directly interact with the EventCloud, which is wrapped, and employ an event-format that enables attachments. An event contains a semantic description, which is the EC event itself, a list of attachment descriptors and the attachments themselves. An attachment

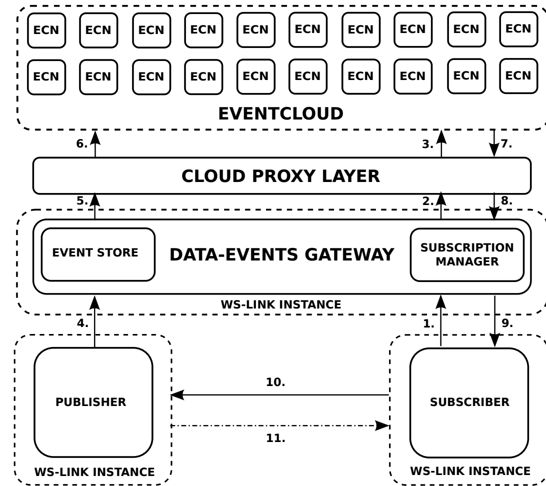


**Figure 1: Proposed middleware architecture**

can be either a resource (e.g. a file) or a data-structure. In the former case, the corresponding descriptor contains information like the name and the mime-type. In the latter case, the descriptor may provide the language-specific type (i.e. the Java fully qualified name of the class, in case the event should be handled only by Java-based subscribers) or a URI to the WSDL document describing such structure. The URI can be employed to enable dynamic instantiation on the subscriber-side.

Publisher and subscribers now interact with *Data-Events Gateways*, that are responsible for decoupling event-descriptions, containing RDF quadruples, from event-data, that is not used for subscription matching. Next section details how a pub/sub interaction takes place in the proposed architecture.

## 3. EXTENDING THE EVENTCLOUD

The architecture described in Figure 1 enables the delivery of messages containing attachments, by means of the *Data-Events Gateway*. This component is able to correlate EC native events and Data-Events. Attachments are never imported into the EC, to prevent efficiency issues due to the storing process.

A simple publish-subscribe scenario can better explain the role of each component inside the architecture. When a service wants to subscribe for a specific kind of event, it interacts with the gateway that exposes a *WS-Notification* interface [3]. The gateway prescribes a specific subscription structure. A subscription can be either topic or content-based: in the first case, the content is a simple string representing the topic; in the second case, a query language can be specified and the content is a query written in this language (currently SPARQL).

The subscription is performed by invoking the *subscribe* operation exposed by the gateway (interaction 1 in Figure 1). The gateway is responsible for collecting all the subscriptions and saving them to a *Subscription Manager* (SM). Then, it subscribes itself to the *EC*, by interacting with the *Cloud Proxy Layer* (CPL) (interactions 2 and 3). The *CPL* enables a high level of flexibility, since it decouples the gateway from the *EC*: if these components are deployed locally

[2]http://www.play-project.eu/

[3]http://www.oasis-open.org/committees/wsn

to each other, the *CPL* will enact interaction 3 by means of a local method invocation; on the contrary, if the *EC* is remotely deployed, the interaction will follow the *WS-Notification* standard. The *EC*, itself, is in charge to handle both topic and content-based subscriptions, by employing a matching algorithm, whose details can be found in [2].

Our first contribution is a two-layered subscription system, where gateways act as filters. When new external subscriptions arrive, the gateway layer stores them, checks if there are other subscriptions for the same topic (or kind of content), by querying the *SM*, and, if not, registers itself to the *EC* layer for the specified type of events. The subscription is finally stored in the *EC*. Despite this work does not address scalability aspects, it is worth noticing that the architecture can scale horizontally, since gateways can be replicated. Consider a scenario including $X$ external subscribers for topic $T$; in this scenario, if an event related to $T$ occurs, it is dispatched to all the subscribers in $O(X)$ messages, if only one gateway has been deployed and a round-robin approach is assumed. Anyway, if $Y$ gateways are deployed and the subscribers are uniformly assigned to all of them, the event is dispatched in $O(X/Y)$ messages, since different gateways can serve different sets of subscribers, in parallel.

When a service raises an event, this will be sent to the gateway by invoking its *notify* operation (4). When the event is received by the gateway, the latter is in charge of creating an identifier for the event, adding it to its description and forwarding the description to the *EC* (5, 6). The whole event is kept inside an *Event Store* (ES), that is local to the gateway. There is no need to move the whole event inside the *EC*, since only its description is used to match existing subscriptions. Once the event description enters the *EC*, it will be inspected to verify if it matches any previous subscription. In case of matching, the *EC* notifies the gateway, by means of the *CPL* (7, 8). The gateway extracts the event identifier from the event description, queries the *ES* to retrieve the whole event and, finally, dispatches it to the actual subscribers, by means of the *SM* (9).

Despite the described architecture avoids attachments transfers from/to the *EC*, attachments are moved from publishers to gateways, and then back to subscribers. To achieve a higher degree of efficiency, attachments could be transferred from publishers, directly to subscribers. Moreover, such optimization should not impact on the design of publishers' and subscribers' business logic. Next section addresses both the efficiency and transparency concerns.

# 4. IMPROVING DATA-TRANSFERS EFFICIENCY

Our main contribution is an architectural solution to enable direct data-transfers from publishers to subscribers. We have designed and realized WS-Link: a Java-based framework that enables configuration-by-exception of data-transfer aspects, for the attributes of those entities that are exchanged during web-services invocations.

It extends the DynO4WS framework, that provides Dynamic Object Offloading capabilities to web-services endpoints, by taking advantage of the Apache CXF[4] message interception API. DynO4WS (Dynamic Offloading for Web-Services) is a middleware aimed at decoupling the semantics of service invocations from the way data is moved between interacting service-endpoints. To this end, the framework allows for the customization of the transfer process of the attributes belonging to entities exchanged during service invocations. Such customization takes place by plugging a *LoadingStrategy* (see Figure 2). This abstract component decides which attributes, inside an entity that is going to be sent as an IN/OUT parameter of a service invocation, should be serialized at once, and which ones should be made available for later access (because they are not likely to be used by the remote endpoint or they exhibit a considerable size). In the latter case, the attributes are made available from an *OffloadingRepository*.

One key feature of the framework is its transparency: developers do not have to change any line of code of the endpoints business logic, thanks to a dynamic proxing system we extensively describe in [21]. When offloading takes place, the framework adds specific meta-data to the header of the outgoing SOAP message concerning which attributes have not been sent yet, and how they can be retrieved. When the framework instance on the remote endpoint receives the message, the *ProxyManager* generates a proxy according to such meta-data, thus hiding the loading strategy.

WS-Link extends our previous work by implementing a specific *Loading Strategy* that enables fine-grained configuration for those attributes that should not be transferred as in a common web service interaction, by means of Java annotations [3]. The customization can take place by means of three key annotations. *@Strategy* allows the developer to specify a component that is in charge of deciding when the marked attribute should be transferred. Basically, an attribute can be transferred when the invocation takes place, or on-demand, when the remote endpoint actually needs its value. Anyway, there can be several reasons to choose between these options and some of them depend on the runtime value of the attribute. For instance, if the size of the attribute is negligible, it could be useless to delay its transfer. The framework lets the designer plug its own strategy and configure it at the attribute level, by means of key-value pairs that can be nested inside the *@Strategy* annotation (using the *@StrategyParam* syntax). Every time an event is raised, the framework retrieves the value of the *@Strategy* annotation, along with the nested key-value pairs, and invokes the $shouldOffload$ method that each strategy must implement. The strategy can dynamically decide if the annotated attribute should be serialized, since at each invocation it receives a map containing the key-value pairs declared inside the *@Strategy* annotation and the event instance. The implementing strategy cannot be selected at runtime, but a strategy could apply different criteria, based on runtime inputs.

*@Consistency* can be used to indicate if it is necessary to keep a serialized copy of the attribute value, as it was at the moment of the invocation, in case its transfer is going to be delayed, according to the strategy. The default value for this parameter is *"MAKE_COPY"*, since it guarantees that the receiving endpoint will obtain the same value as if no delayed transfers took place. If the attribute is not likely to change during the period between the service invocation and the attribute access, a *"KEEP_REFERENCE"* consistency value can be employed; in this case, a pointer to the attribute is kept until it is requested or the entity it belongs to gets garbage-collected on the receiver side. This last aspect is handled by the DynO4WS framework, by creating

[4]http://cxf.apache.org/

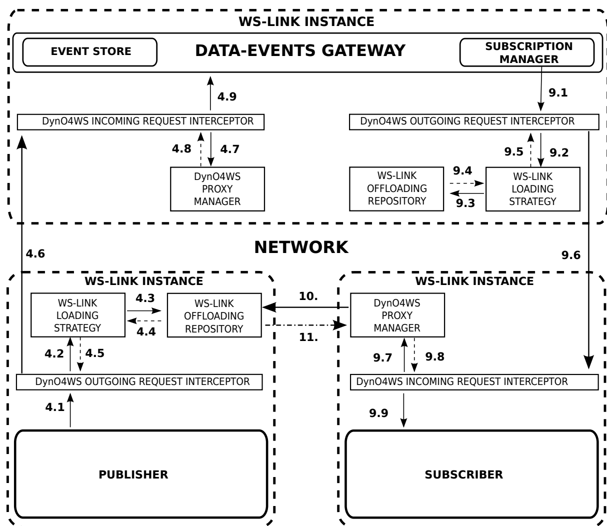


**Figure 2: Detailed architectural view explaining how to enable transparent direct pub/sub data-transfers, by means of WS-Link.**

**Listing 1: WS-Link based configuration of attachments in the Event class**

```
@Strategy(impl = PureLazyStrategy.class)
@Consistency(ConsistencyType.KEEP_REFERENCE)
@Encoding(EncodingType.XML)
private byte[] attachments;
```

dynamic proxies for the exchanged entities and intercepting their garbage-collection. Finally, the *@Encoding* annotation allows for the customization of the serialization format (currently XML or JSON) as well as the possibility to enable message compression.

It is worth to explain the role of the WS-Link framework inside the architecture depicted in Figure 1. For this purpose, Figure 2 gives more details about how we enable direct data-transfers. When the publisher raises an event, this is handled by the WS-Link framework, before it is sent to the gateway. In order to delay their transfer, we have configured the *attachments* attribute of our Event class, as shown in Listing 1.

The WS-Link framework detects the annotated attributes[5] on the publisher-side and enacts the offloading process, according to the value of the declared annotations. Since the attachments should be transferred on-demand, WS-Link saves them in a local repository [6] and adds meta-data to the outgoing SOAP message, indicating how these attachments can be retrieved (interactions from 4.1 to 4.5).

When the event reaches the gateway (4.6), another instance of WS-Link decodes the SOAP message and instantiates a dynamic proxy that hides the "laziness" of the attachments transfer (4.7 to 4.9). This proxy holds the remote references to attributes and is kept until the event gets delivered to all the subscribers. The gateway does not need to inspect the attachments, so it will never trigger an attachment transfer. Before the gateway sends the event to a subscriber, WS-Link copies the meta-data inside the proxy to the outgoing SOAP message (9.1 to 9.5).

Listing 2 details how remote references are encoded into SOAP messages. The SOAP header refers to the invocation of *notify* performed by the gateway. Line 2 shows both the id of the current call and the one related to the message sender. Lines from 3 to 10 show how the event attachments can be retrieved. Attribute *cid* at line 4 refers to the call that generated the *attachments* attribute (the invocation of *notify* performed by the publisher). Attribute *host* at line 5 indicates the endpoint of the repository hosting the attachments. Finally, the id at line 7 prescribes how to query such repository, to retrieve the desired element.

The meta-data is finally decoded by the WS-Link instance at the subscriber-side. This process, whose details can be found in [22], is called *"Attribute Loading Delegation"*; in this case, the gateway delegates the publisher's repository to make the attachments available for subscribers.

**Listing 2: SOAP header containing metadata for proxy instantiation, produced when the Gateway invokes the notify operation exposed by the subscriber.**

```
1  <soap:Header>
2   <wslink cid="14967149050078" nid="Gateway">
3    <param name="0">
4     <field cid="14964341660014"
5      host="http://10.0.0.9:9999/DynO4WS/repo?wsdl"
6      name="attachments">
7      <id>{nid:Pub,cid:14964341660014,param:0,
           name:attachments}
8      </id>
9     </field>
10   </param>
11   ...
12  </wslink>
13 </soap:Header>
```

Once the event is delivered to the subscriber (9.6), its WS-Link instance creates a new proxy, that is able to retrieve the attachments, directly from the publisher (9.7 to 9.9). A typical subscriber-side logic should inspect the description of the event, in order to understand why such event arrived (e.g. by checking the topic). Subsequently, it should check the attachment descriptors, to decide if it is able to handle the attachments, and possibly inspect the attachments themselves. Each of these attributes can be accessed by invoking the corresponding getter method of the event class. When a getter method is invoked on a proxy generated by the *Proxy Manager*, the latter checks if the value has been already transferred to the local endpoint; if not, it triggers an invocation to a remote repository, according to the meta-data attached to the incoming SOAP message. In our case, when the *getAttachments* method, inside the event class, is invoked, the *Proxy Manager* retrieves the attachments from the repository at the publisher-side (10, 11). Thanks to the dynamic proxing system, both publisher's and subscriber's business logic can take advantage of delegation, without modifying any line of code. The WS-Link framework transparently avoids useless transfers, thus increasing the efficiency of our pub/sub system, as proven in the following evaluation.

[5]In this phase we group the attachments together in a single attribute, but more attributes can be added, by extending the Event class, to create custom events where each attachment can be managed in a different way.

[6]The "local" term should not be intended in a strict way. By default, the repository is deployed locally, but it can be shared among several remote instances of WS-Link, in order to take advantage of network topology and to keep the attachments available, if the publisher goes offline.

## 5. PERFORMANCE EVALUATION

In order to evaluate our solution, we have realized a prototype based on the EventCloud middleware and the WS-Link framework. The evaluation process has been organized into two phases. In the first phase, we have measured the average time, needed to deliver an event containing an attachment, if no attribute loading delegation is performed. In such scenario, the event is completely transferred from the publisher to a *Data-Events Gateway* and eventually to the subscriber.

In the second phase, we enable attribute loading delegation, by means of the WS-Link framework, in order to avoid useless data-transfers. In this case, the attachment is "offloaded" to a publisher's local repository and eventually moved directly to the subscriber. In a real scenario, subscribers may access or not the event attachments. Our solution can take advantage of this behaviour, since a data-transfer is triggered only if required by the subscriber. However, in our tests we consider a situation where subscribers always access the attachments of the event, since we want to evaluate the overhead added by our middleware in the worst case.

The process has been repeated for three kinds of events, respectively carrying 1MB, 10MB and 100MB-sized attachments, in order to simulate different media like images, documents and videos.

Considering that the offloading process adds a computational overhead, as documented in [22], we have evaluated our solution for three different network throughputs (10Mb/s, 100Mb/s, 1Gb/s), in order to verify what are the network conditions under which such overhead may become unacceptable. Finally, the tests have been run on three nodes, respectively hosting the publisher process, the gateway and the subscriber, equipped with a Intel Xeon E5520 CPU @ 2.27GHz and 12 GB of DDR3 memory.

Figure 3(a) shows the average delivery time for an event carrying a 1MB attachment. By avoiding useless data-transfers the WS-Link framework can lead to an improvement of 53.8%, for the 10Mb/s throughput test, 56.1%, for the 100Mb/s test and 56.5% for the 1Gb/s one. The 10MB attachments tests results (Figure 3(b)) show that performances decrease if a higher throughput network is considered (respectively 45%, 28.7% and 18.2% of improvement for the considered throughputs). This behaviour is confirmed in the last round of tests (Figure 3(c)), where an improvement of 47% is achieved for a 10Mb/s throughput network, but a worse result (27.5% for the 100Mb/s test and -11.4% for the 1Gb/s one) is obtained when high-throughput networks are considered.

These anomalies are due to a still not mature implementation of the un/marshaling components employed by the WS-Link framework. In facts, our tests show that encouraging performances can be achieved, if the serialization time is negligible compared to transfer one[7]. We are confident that better results can be achieved by adapting the CXF serialization components to our needs. Moreover, the 10Mb/s throughput tests show that our current implementation is already competitive for internet-based scenarios [8].

[7]In the 100MB attachment tests, the serialization time corresponds to 8.62% of the total delivery time, when a 10Mb/s network throughput is considered, 46.39% for a 100Mb/s throughput and 82.52% for a 1Gb/s one.

[8]The considered 10Mb/s throughput corresponds to the average Internet speed at the time we are realizing this work (http://www.netindex.com/)

## 6. RELATED WORK

Merging EDA, semantic event processing and SOA is gaining increased attention from researchers and industry. Many works are focusing on the combination between the pub/sub paradigm and semantic technologies [8, 7] while others focus on the adoption of the pub/sub paradigm itself for web services [6, 18]. Eugester *et al.* [6] give a comprehensive survey about the pub/sub communication paradigm. Indeed, most of the efforts concentrate more on the definition of protocols rather than on improving the efficiency of data exchanges.

In [18], the authors propose to combine pub/sub and Web-Services to overcome the request/response model. To carry out a data-transfer, the publisher notifies the subscriber about the availability of data to be retrieved through a further and explicit request/response interaction. In our work, we keep the request-response mechanism, but we make it completely transparent, by means of dynamic proxies. In [19] the authors provide a system based on their *DSProxy* as a cross-platform solution. It provides store-and-forward capability to SOAP messages, by applying compression of SOAP and XML and facilitating the traversal of multiple heterogeneous networks. Despite the authors try to offer a lightweight Web-Service based standard solution, they do not address huge volumes of SOAP messages or attachments, as addressed by our solution.

The authors in [10] offer an experimental pub/sub infrastructure to be used with Web-Services enabled applications. They demonstrate that the performance gap between traditional event-based technologies and the Web-Services based approach is not necessarily significant. They emphasize the decoupling characteristics of the Web-Services and event-based design, and they propose a pub/sub infrastructure compliant with the WS-Notification standard. In any case, this solution does not take into account the nature of the data exchanged between services by means of the pub/-sub mechanism. This aspect is extensively addressed in our work.

Nevertheless, how to deliver huge data among Web-Services by means of pub/sub, remains an unanswered question in the community. Data-centric pub/sub has been addressed by the *Data Distribution Service (DDS)* OMG standard [14] and by some related research papers [16, 12]. The purpose of the specification is to provide a common application level interface that defines the data-distribution service. However, the DDS approach requires the data to be transferred to a global data space. Managing global data might introduce some performance degradation due to distribution unawareness. This problem should be addressed by optimizing the distribution of the global space according to data consumption, similarly to the approach in [20], where the authors propose to explicitely characterize (micro)objects with some non-functional properties that guide the deployment in a distributed environment. Our solution employs a global space only for event descriptions, while the attachments store can be distributed, to take advantage of network topology. Moreover, we exploit declarative statements to inform the underlying middleware about the specific semantics that characterize the interaction, in order to improve performance and scalability.

## 7. CONCLUSION

There is an increasing need for business and communication flexibility, that can be achieved by combining Service

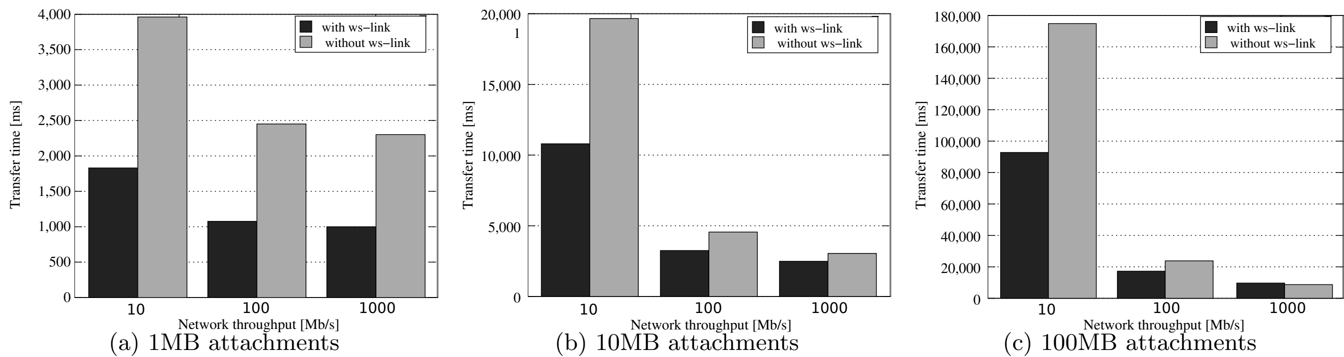


**Figure 3: Comparison of average event-delivery time with and without attribute loading delegation**

Computing and Event-Driven Architectures. In this context, it is crucial to provide solutions to efficiently exchange large amounts of data. We have presented a middleware, integrating the WS-Link framework and the EventCloud infrastructure, that allows for efficient, data-intensive, event-driven communication. Our evaluation shows that we can reduce the average event delivery time, compared to a fully mediated solution, but we are planning to improve our results, by optimizing the serialization process for large attachments. Moreover, we want to verify the degree of scalability characterizing our platform.

The flexibility of the event-driven paradigm (and pub/-sub communication model) has been successfully applied to scientific workflows, as described in [13, 1]. Anyway, these works focus on the paradigm itself and do not address large data exchanges. Therefore, our next step will be the application of our solution to achieve a higher level of efficiency for data-intensive scientific workflows.

Finally, we want to add autonomicity features to our middleware, so that repositories can be shared, replicated or migrated, based on runtime workload, in order to improve scalability, availability and fault-tolerance.

### *Acknowledgements*

This work is supported by Province of Benevento, EU FP7 STREP project PLAY and French ANR project SocEDA.

## 8. REFERENCES

[1] A. Alqaoud, I. Taylor, and A. Jones. Scientific workflow interoperability framework. *International Journal of Business Process Integration and Management*, 5(1):93–105, 2010.

[2] F. Baude, F. Bongiovanni, L. Pellegrino, and V. Quema. D2.1 requirements eventcloud. Technical report, European Commission, 2011. Project Deliverable PLAY.

[3] J. Bloch. JSR 175: A metadata facility for the Java programming language. `http://jcp.org/en/jsr/detail?id=175`, Sept. 30, 2004.

[4] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[5] T. Erl. *SOA Design Patterns*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2009.

[6] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, June 2003.

[7] F. Facca, S. Komazec, and M. Zaremba. Towards a semantic enabled middleware for publish/subscribe applications. In *Semantic Computing, 2008 IEEE International Conference on*, pages 498 –503, aug. 2008.

[8] I. Filali, F. Bongiovanni, F. Huet, and F. Baude. A survey of structured p2p systems for rdf data storage and retrieval. *Transactions on Large-Scale Data-and Knowledge-Centered Systems III*, pages 20–55, 2011.

[9] G. Hohpe and B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[10] B. Kowalewski, M. Bubak, and B. Baliś. An event-based approach to reducing coupling in large-scale applications. In *Proceedings of the 8th international conference on Computational Science, Part III*, ICCS '08, pages 358–367, Berlin, Heidelberg, 2008. Springer-Verlag.

[11] O. Lassila, R. Swick, et al. Resource description framework (rdf) model and syntax specification, 1998.

[12] C. Lee, J. Hwang, J. Lee, C. Ahn, B. Suh, D.-H. Shin, Y. Nah, and D.-H. Kim. Self-describing and data propagation model for data distribution service. In *Software Technologies for Embedded and Ubiquitous Systems*, volume 5287 of *LNCS*, pages 102–113. Springer Berlin / Heidelberg, 2008.

[13] G. Li, V. Muthusamy, H.-A. Jacobsen, and S. Mankovski. Decentralized execution of event-driven scientific workflows. In *Services Computing Workshops, 2006. SCW '06. IEEE*, pages 73 –82, sept. 2006.

[14] G. Pardo-Castellote. Omg data-distribution service: Architectural overview. In *ICDCSW*, page 200, 2003.

[15] E. Prud'Hommeaux and A. Seaborne. Sparql query language for rdf. *W3C working draft*, 4(January), 2008.

[16] M. Ryll and S. Ratchev. Towards a publish / subscribe control architecture for precision assembly with the data distribution service. In *Micro-Assembly Technologies and Applications*, volume 260 of *IFIP International Federation for Information Processing*, pages 359–369. Springer Boston, 2008.

[17] A. Shalloway and J. Trott. *Design patterns explained : a new perspective on object-oriented design*. Addison-Wesley, Boston, Mass., 2004.

[18] I. Silva-Lepe, M. Ward, and F. Curbera. Integrating web services and messaging. In *Web Services, 2006. ICWS '06. International Conference on*, pages 111 –118, sept. 2006.

[19] E. Skjervold, T. Hafsø ande, F. Johnsen, and K. Lund. Enabling publish/subscribe with cots web services across heterogeneous networks. In *Web Services (ICWS), 2010 IEEE International Conference on*, pages 660 –668, july 2010.

[20] J.-M. S. Wams and M. van Steen. Simplified distributed programming with micro objects. In *FOCLASA*, pages 1–15, 2010.

[21] Q. Zagarese, G. Canfora, and E. Zimeo. Dynamic object offloading in web services. In *Proceedings of SOCA and RTSOAA, KASTLES, LCPS*, pages 58–65, 2011.

[22] Q. Zagarese, G. Canfora, E. Zimeo, and F. Baude. Enabling advanced loading strategies for data intensive web services. In *ICWS*, 2012.