Evolving a Core Banking Enterprise Architecture
Leveraging Business Events Exploitation

Beatriz San Miguel, Jose M. del Alamo and Juan C. Yelmo
Center for Open Middleware (COM), Universidad Politécnica de Madrid (UPM), Campus de Montegancedo, E-28223, Pozuelo de Alarcón, Madrid, Spain

Keywords: Event-Driven Architecture (EDA), Enterprise Architecture (EA), Event, Middleware, Service Oriented Architecture (SOA).

Abstract: Business information has become a critical asset for companies and it has even more value when obtained and exploited in real time. This paper analyses how to integrate this information into an existing banking Enterprise Architecture, following an event-driven approach, and entails the study of three main issues: the definition of business events, the specification of a reference architecture, which identifies the specific integration points, and the description of a governance approach to manage the new elements. All the proposed solutions have been validated with a proof-of-concept test bed in an open source environment. It is based on a case study of the banking sector that allows an operational validation to be carried out, as well as ensuring compliance with non-functional requirements. We have focused these requirements on performance.

1 INTRODUCTION

In 2011, Santander Bank (together with its technological and operational divisions ISBAN and PRODUBAN) and Universidad Politécnica de Madrid created a joint technology center called Center for Open Middleware (COM). COM is the incubator of an open software ecosystem aiming at developing middleware solutions and experimenting with new software architecture approaches.

There are different technologies hosted under the COM umbrella and one of the key ones is an Enterprise Architecture (EA) called BankSphere (BKS). Created by the Santander Group, BKS is a set of integrated design tools, and a deployment and runtime environment that speeds up the development of new bank software such as applications for customers, call center staff or bank branch workers. BKS has constantly evolved to fulfil Santander requirements, and now it is required to enhance the generation and exploitation of real time business information. This last part can be achieved by applying an event-driven approach in BKS.

Event Driven Architecture (EDA) allows systems and applications to deliver and respond to real time information, helping to support business needs from an IT management standpoint (Malekzadeh, 2010). Thus, it has associated both technological benefits and business advantages. As regards the former, EDA provides loose coupling between its components, which reduces dependencies and allows modifications without giving rise to side effects. A many-to-many communication pattern is also applied, facilitating the reusability of information and the freedom to act independently with the received information. All the above creates an adaptive and flexible architecture that results in business advantages. EDA enables faster, more agile and more responsive business processes, enhancing the informed decision making model and the automation and motoring of operational activities, among other business advantages.

In early 2012, we started to work on a pilot project intended to research, analyse and evaluate event-oriented approaches, architectures, tools and technologies and its potential application and integration into the context of Santander Group architectures. Specifically, the project focuses on the correct incorporation and use of real time business events in the BKS context, identifying the key necessary elements and integrating them into BKS, while minimizing interference with the existing architecture and procedures. Moreover, a requirement to all COM solutions is that they must be based on open-source technologies that open up new possibilities.

The specific requirements necessary to evolve the core banking EA towards an EDA approach are: the definition of business events; the design of a reference architecture, which identifies the integration points with the specific EA, and the description of the initial governance approach to manage the new elements.

This paper is organized as follows. First, section 2 covers related work and puts our work in perspective. Section 3 gives an overview of the background of the project: EDA and BKS main concepts. Then, a proposed solution to introduce EDA in BKS is presented in section 4. Section 5 includes an operational validation though a case study and a non-functional validation focussing on performance. Finally, section 6 concludes the paper and introduces areas of future research.

2 RELATED WORK

Diverse studies have tackled the introduction of business events in existing architectures of different domains. Most of them describe general approaches for EDA and SOA integration such as (Taylor et al., 2009) or (Malekzadeh, 2010), while others address only specific issues of the EDA integration like modelling, simulation, methodologies, performance, etc. For example, (Clark and Barn, 2012) proposes an EDA modelling notation and its associated simulation language; (Weigand, 2011) describes unified event ontology and a methodology for event-driven business processes; and (Vidačković et al., 2010) explains a business-oriented development methodology for event processing.

The papers reviewed provide the theoretical basis to evolve EAs. Most of them include a validation exercise through a case study in the application field. However, they are usually academic or simplified examples, not practical experiences for real-world EAs, since most companies, and banks in particular, do not usually publish them. Our contributions are focused on this last point, a real-world EA evolution and its associated solutions, which we think are of the utmost interest for engineers and practitioners.

3 BACKGROUND

3.1 Event Driven Architecture

EDA is a software architecture style based on multiple entities communicating asynchronously via announcements or notifications, known as events. Instead of the traditional synchronous, request-response interaction model, where a requestor asks for services or messages and waits for an answer from a replier; in EDA, events are transmitted in a fire-and-forget mode. In other words, events are communicated without a previous request and without being concerned about what happens afterwards with them.

Basically, an event is a change in a state within a particular system or domain that merits attention from other systems (Taylor et al., 2009). The term has been given other meanings, depending on the context. It can refer to the actual occurrences (the things that have happened), which are also known as instances of a particular type of events. On the other hand, we can use '*event*' or '*notification*' to specify the particular communication of an event instance. Generally, the word '*event*' is used in both cases without distinction. We will use '*event instance*' or '*event notification'* where its distinction is relevant.

We can think about different types of event taking place in a company, such as events related to low-level technical information, software activity, user actions or business data. Furthermore, we may also consider events happening outside the company (e.g. stock exchange markets, social networks or any other data sources). By way of example, low-level technical events can be information from sensors, ATM status, network data or activity in many other devices. Software events can indicate calls to methods, execution of services or exceptions in the execution of a program or a process. We may understand user events as actions or information generated by both customers and workers of a company. Finally, this paper focuses on business events. They are those generated by the core company activities and represent relevant information that has impact on its economic development and management. For instance, in a financial institution, business events can derive from the registration of new customers, canceling of services, money withdrawals, or the contracting of products such as credits, mortgages, etc.

A generic EDA is made up of three core layers: producers, channel and consumers (Figure 1). The process begins at the producer layer, detecting, creating and sending events through a channel, and ends when consumers receive these event notifications and carry out a specific task (automatically or with human intervention).

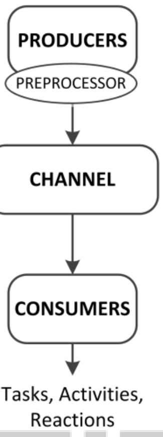

Figure 1: Generic EDA layers.

Producers can contain a software subcomponent called preprocessor to add intelligence to the event publication. It can carry out different tasks such as, filter, prioritize or homogenize the produced events. Thus, only the most relevant events will be sent or all event notifications will have the same format.

The channel is responsible for transporting event notifications between producers and all associated consumers. It usually takes the form of a Message Oriented Middleware (MOM), which is a software infrastructure that can send and receive messages between distributed systems, regardless of platforms, technologies and resources used.

A MOM can use different messaging models such as point-to-point or publish/subscribe. In the first model, only one consumer receives a particular event notification, while in the second, more than one consumer may express their interest in a set or subset of event types, in order to be notified when a producer generates their registered interest (Eugster et al., 2003). Technically, it is usually achieved thanks to an intermediary entity known as broker that receives all event notifications from producers and routes them to the subscribed consumers, using queues that store event notifications if necessary.

Consumers can be any entity such as software components, applications or systems that react to the received notifications. For example, it can create new event notifications, invoke a service, initialize a business process, increase a value or notify humans to carry out manual tasks. There is a special kind of consumer that is known as an event processing engine, which encompasses the set of computational procedures to carries out operations with events such as reading, creation, transformation, deletion or correlation (Etzion and Niblett, 2010). Because of its importance, it is frequently considered as an independent layer in EDA.

There can be three styles of event processing which may be used together (Michelson, 2006): simple, stream and complex. The former is the most basic process: an event is received and it produces an action. On the other hand, Event Stream Processing (ESP) continually receives all kinds of events (ordinary and notable) and through established rules or queries on the flow of data, and then decides whether or not to forward events (or information about them) to other consumers. Finally, Complex Event Processing (CEP) relates different event types from various sources to produce new events or extract relevant information.

3.2 BKS Banking Services Platform

BKS is a set of development tools created by the Santander Group that allows programmers to create new banking applications quickly. It includes a design framework integrated with the Eclipse IDE, and a deployment and runtime environment based on Java Enterprise solutions and web technologies.

BKS has been designed to allow the reuse of software components and simple, fast programming. The former is achieved by its Service Oriented Architecture (SOA)-like approach, where pieces of software are developed and exposed to be reused by other components. The latter is carried out by using a visual programming environment that allows programmers to design applications through usable graphical user interfaces (GUI). It hides the code details and lets programmers to use graphic symbols that represent software components.

Simply put, BKS programmers can create presentation and business flows by reusing previously implemented software components. The business flows are exposed by a *facade* and can be used to create banking applications. An application is usually constituted by a main presentation flow that calls different business flows, which in turn call backend operations or services.

A BKS application is typically turned into a Java Enterprise Edition application, exposing the application through a web module (WAR) and implementing the business logic in various Enterprise JavaBeans (EJB). It is then executed in a runtime environment provided by BKS (Figure 2).

The execution of BKS applications at runtime is as follows. First, a request for an application is detected and redirected to the operation container. It

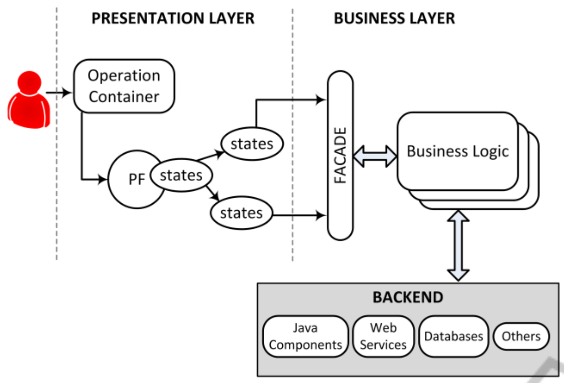

Figure 2: Basis of BKS applications at runtime.

invokes different initial operations and the states defined by the presentation flow (PF). These states can call business flows through a common facade. At this point, the execution entails invoking backend operations and external components defined by the business logic. Finally, the request ends when all presentation states have been executed and the control is returned to the user.

4 BKS MEETS EDA

In the previous sections we have reviewed the foundations of EDA and the main features of BKS. Converging on a solution that brings the best of these architectures requires extending current BKS capabilities and identifying the key integration points that interfere minimally in the existing architecture and the associated procedures.

Specifically, we include the results obtained for evolving an EA towards the EDA paradigm in the following subsections. The specific results are: the definition of the new banking business events, the design of a reference architecture to integrate EDA that allows business event generation and exploitation and identifies the specific integration points with BKS, and finally, a description of the initial governance approach to manage the new EDA elements.

4.1 Business Event Definition

The event definition entails deciding which semantic and data each event instance must contain, and which data-exchange format is assigned to event notifications. Given that there is no standard or a generally adopted event format, and there is a huge variety of business event types with different meanings and aims, the event definition is one of the most problematic issues in EDA integration.

We have carried out a study with the possible alternatives that are used or can be used to communicate and later, process business events, concluding the following:

- There is a wide range of event definitions in different formats that addresses specific issues in a company and changes depending on the purpose of the event, the domain or the business layer (Becker et al., 2012).
- Event notifications can be implemented by any data-exchange format such as XML, JSON, Google Protocol Buffers, CSV, ASN.1 or Hessian (Aihkisalo, 2011) and (Maeda, 2012).
- Event processing engines can use Java Objects, expressions or JSON-based or tag-delimited languages to represent events.

Moreover, we have examined some initiatives proposed in the field of web services, such as the WS-EventDescription or the WS-Notification. They are not specific event definitions and cover the description of communication protocols between web services. Events in other domains, for example the specification Activity Streams for social web application, have also been reviewed.

Clearly, there is no single solution to choosing language and some of them can be used either jointly or separately. However, there is a trend towards formats that allow the inclusion of two differentiated parts: header and body (Michelson, 2006) and (Etzion and Niblett, 2010). The header includes metadata information: generic event information such as the name, identifier, occurrence time or producer identification, or can describe the event type. On the other hand, the body or payload contains the specific data on the event instance.

We have decided to follow the aforementioned structure in an XML format. We have defined a general XML Schema Definition (XSD) that contains the basic structure for all event types. Here, the header has three basic elements common to all events:

- *eventType*: indicates the kind of event according to the hierarchy of the Santander Group's business event catalogue. It has an attribute denominated as a category that specifies the nature or domain of the event and the value is a text string that contains two parts separated by a dot, indicating the business area and the specific type.
- *createdTime*: contains the timestamp of an event occurrence.
- *createdBy*: identifies the event producer that generates the event.

The body is limited by an element called *recordAppData* that contains a reference to other separated XSD. There is a XSD for each event type and each of them describes the specific data of an event type. It is important to highlight that we have divided the header and body definitions in different XSDs to allow specialized actors to handle a specific part.

4.2 Reference Architecture

An EDA process starts with the detection and creation of events. Although BKS does not include any EDA layers, it uses relevant business information that can be mapped to banking business events. As a result of these facts, event producers have been detected as the only integration point between EDA and BKS architectures (Figure 3).

We have detected two main ways to incorporate event producers in BKS applications:

- Explicit way. BKS programmers must include a call to an event producer component to generate business events, configuring the exact values for the business event instance. In other words, they have to decide where to include the creation of the event inside the business logic and moreover, obtain the context data that corresponds to event instances.
  This solution has several drawbacks. First, programmers have to acquire new responsibilities and understand new concepts related to business areas that differ from their daily technical work. Secondly, this incorporation in existing banking applications, which are currently in production, implies their modification and it can entail risks in stable applications.
- Implicit way. Here, the incorporation of event producers is almost transparent for programmers. They are strictly limited to defining the business logic and the business event generation is associated with calls to business flows.
  The main disadvantage of this solution is the low quality of the generated events. They correspond to calls to functional business methods but they do not necessarily tie in with business event definitions. To solve this last point, a preprocessor can be included. It can create real business events based on execution traces obtained from calls to the facade. However, it requires an in-depth analysis of the context and each functional component to be related to business events.

Both alternatives can coexist in BKS. We have specifically proposed to use explicit business event generation for new BKS applications and implicit generation for existing applications.

The rest of the EDA layers (channel and consumers) will be new elements in BKS. At present, BKS already incorporates a stable commercial messaging system for logging the application. It allows execution traces to be stored in a database to be batch processed. Therefore, a MOM that allows the publish/subscribe model to distribute business events in real time has to be incorporated.

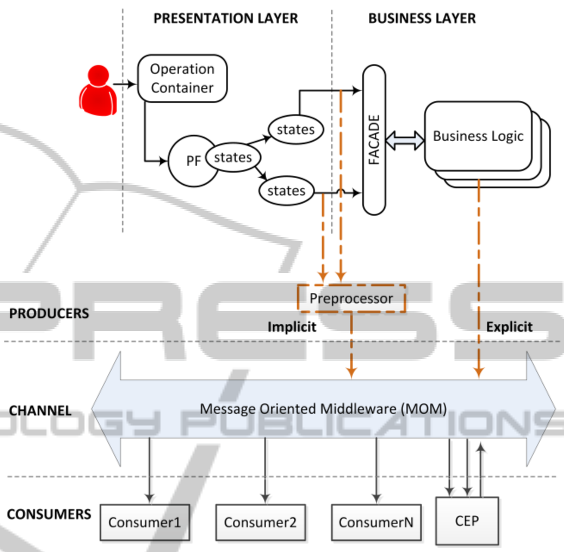

Figure 3: EDA integration with BKS.

4.3 Initial Governance Approach

The incorporation of EDA elements in BKS entails the creation of new operational and organizational processes that allow the Santander Bank to govern business events. Governance is a wide discipline that can be applied on multiple perspectives of a company such as that related to EA, IT, data, business or SOA. Basically, it seeks to define a global structure for establishing and ensuring how the company resources sustain and extend the organization strategies. To begin with, we have identified the organizational processes involved in the creation, use and reuse of business events in the BKS context. This new process has been called event lifecycle and has been defined according to the existing procedures in the bank.

The event lifecycle (Figure 4) describes in design time the different states that must be carried out to define, incorporate and use business events generated by applications. In previous sections, we noticed that BKS programmers know the functional specification and logical model of their applications. However, they ignore the business value of their components. Consequently, other stakeholders must participate.

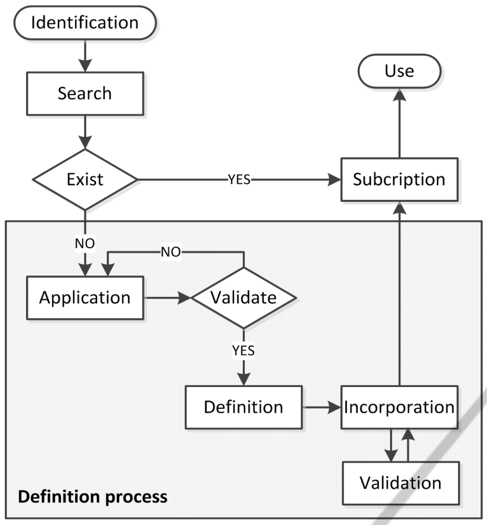

Figure 4: Event lifecycle.

We have identified four main actors in the event lifecycle: a project leader of an event-oriented application that wants to use a specific event, a project leader responsible for the BKS application that generates the event and the corresponding programmers of each project. Members of the quality department participate in approving different stages.

The lifecycle starts with an identification stage: the event-oriented project leader detects the need to consume a specific business event to take advantage of it and examines whether the specific banking business event exists. If it is already incorporated in any BKS application, a subscription to this event is made. Otherwise the new event is defined.

The definition process begins with a formal request for the incorporation of a new business event (application stage). The request goes through a validation stage and if approved, the event is defined and incorporated into a BKS application. The quality department validates this last step again. The definition process ends with the event subscription and use.

5 VALIDATION

In order to validate the previous solutions, we have developed a proof-of-concept test bed in an open source environment. It incorporates several event producers, a MOM, a CEP, and different event consumers that give shape to a banking case study. The case study supports the operational validation of our contributions as well as the evaluation of some non-functional aspects, such as performance.

5.1 Case Study

The case study consists of a wire transfer scenario whose aim is to demonstrate the feasibility of the proposed solutions and the value added to the business by EDA. The scenario takes into account Santander customers who are sending and receiving money from the same bank or others. It must incorporate the technologies and mechanisms that allow the detection, distribution and use of the associated business events. Moreover, it must show any of the multiple possibilities for exploiting these events in real time.

We have identified two main business event types: sent wire transfer and received wire transfer. The former represents events of orders that Santander customers carry out to transfer a certain amount of money to other bank institution. On the other hand, received wire transfers are orders from customers of other banks to Santander customers. Each event type contains the specified header and the following information in the body: session identification, IP address, source account, target account and amount of the transfer.

Our scenario includes the following logical entities (Figure 5):

- Two event producers. Each of them generates a different business event for wire transfers and publishes it through a MOM.
- One MOM that distributes the received events to all the associated consumers.
- A CEP that acts as a consumer and receives all the previous events. It extracts relevant information and displays it in a visualization dashboard. Moreover, if applicable, it generates a new event type that indicates that an individual (not a corporate entity) has received a wire transfer above a threshold. This new event type is called user alert.
- An application displaying a wire transfer dashboard that shows relevant information about the business events of sent and received wire transfers.
- Two consumers that react to the user alert event. One of them is a simulation of a Customer Relationship Management (CRM) that displays records of the wire transfers received by Santander users. Moreover, it can manage, assign and create alerts to call center software with the aim of carrying out commercial actions. The other consumer is a user notification system that sends mails and/or Short Message Service (SMS) to Santander users that have activated the real time notification service to be informed about their transactions.

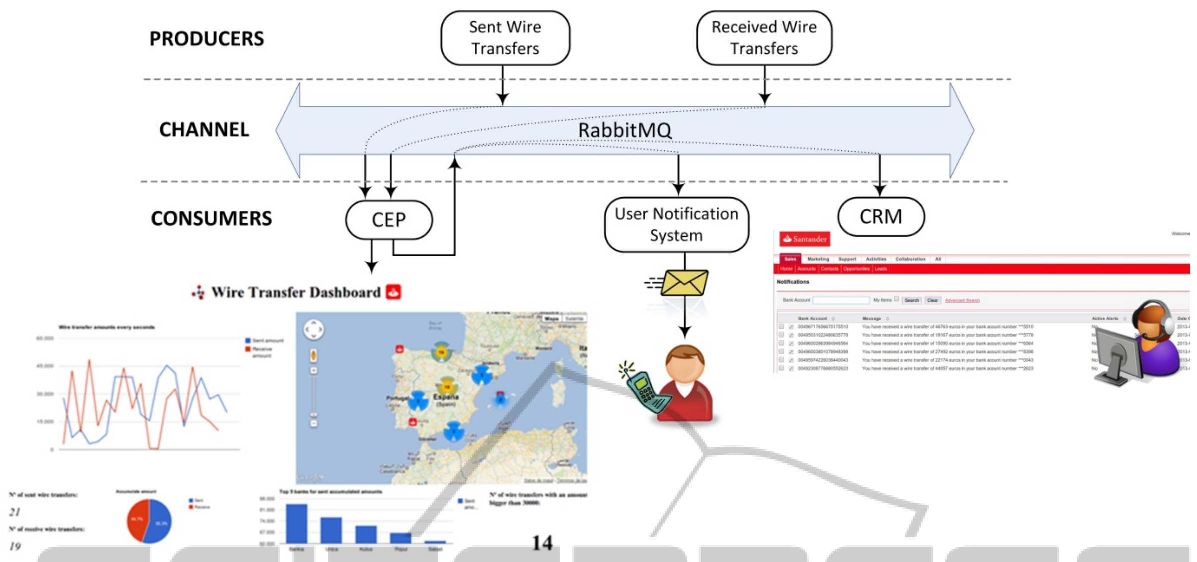

Figure 5: Architecture of the case study.

We have developed the previous components using open source tools and applications, and Java programming language. Event producers have been entirely implemented as Java preprocessors that use information from different BKS applications. They generate event notifications based on the proposed XML event definition for sent and received wire transfers and publish them through a channel.

The channel is implemented by an open source MOM called RabbitMQ (RabbitMQ, 2013) that supports the Advanced Message Queuing Protocol (AMQP) in its version 0.9.1. AMQP covers different messaging models such as publish/subscribe. Its main advantage is to be interoperable allowing consumers and producers to use any programming language or data format.

As regards the CEP consumer, we have selected an open source solution available for Java as Esper, and for .NET as NEsper (EsperTech, 2013). It allows large volumes of events to be processed by applying ESP and CEP. Basically, Esper allows applications to store queries and run the event streams through to obtain information or new event streams. Queries are written using Event Processing Language (EPL) that is similar to the Structure Query Language (SQL) of databases. The differences are: the queries are carried out through an event stream, the data available for querying inside the stream is defined by views and the basic units of data are events and their specific information.

We have obtained information on the received events with EPL queries such as the total number of sent or received transfers, the accumulated amount or the amounts per second for each event type. We also have drawn up a ranking of the top five banks for the most sent accumulated amounts. We extract extra information on the received events by carrying out an enrichment process. Thus, we have located the source of the transfers. In particular, the associated Santander bank branches that are source or destination of transfers.

All the previous information has been displayed in a web application called Wire Transfer Dashboard. We have implemented it using the Google Charts that allow a lot of chart types to be incorporated such as maps, tables, or line or column diagrams. We used WebSocket technology to communicate Esper with the web application.

Finally, we have used Esper to obtain a new event stream with all the received wire transfers that have an amount of more than €3,000 and whose target account belongs to an individual, not a corporate entity. We have developed two consumers that react to it: CRM and the user notification system, implemented by Java web applications.

5.2 Performance Evaluation

We have conducted a first set of performance measures to evaluate whether the selected open source tools fulfil the basic requirements for our case study. In particular, we began by verifying how many messages per sec (throughput) RabbitMQ supports and how many delays experienced by the messages (latency) is added.

We considered a real time scenario, where one producer publishes events and one consumer reacts to them as soon as they arrive. We also use two types of messages: persistent (guaranteeing that it will not be lost) and non-persistent (the opposite case). We sent ten thousand messages of 100 different payload sizes, ranging from 181 to 12KB with a 120B interval. Moreover, each test was run three times and the measures obtained were averaged.

We have carried out the evaluation in two different environments to compare the results. First we used two virtual machines, one for the messaging broker and another for the consumer and producer. Each machine has an Intel Xeon E5520 @2.27 GHz x 4 cores processor, running 2GB RAM, 30 GB disk capacity and Ubuntu 12.04 LTS server 64 bits. Then, we used two physical machines with greater features, a processor Intel Core i7-3720Q @2.60 GHz x 8 cores, 16GB RAM and 80 GB disk capacity, running in an isolated network.

Figure 6 summarizes the results for the throughput of producers, which are very similar but a little lower for consumers. We observe that there is a decrease in the throughput when the message size increases. Also, there is a notable difference between using virtual machines and not using them.

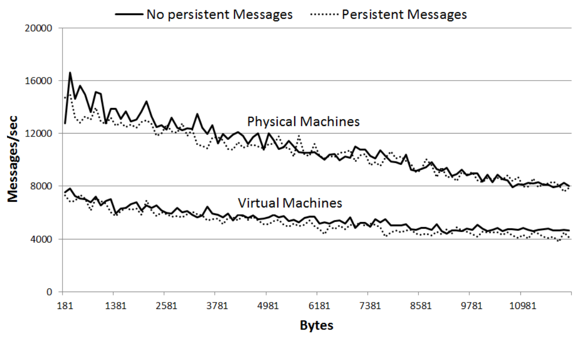

Figure 6: Throughput for producers.

We estimated that event notifications have a size between 2KB to 5KB, where we observe that the throughput is greater than 4,000 messages per sec in both cases (virtual and physical machines), doubling it in physical machines. The results are within the target range and therefore, suggesting that RabbitMQ can be applied in BKS. Moreover, using physical machines we will cover larger amounts of events. Regarding the latency, we obtained an average delay of 119-125 milliseconds with physical machines and 148-158 milliseconds with virtual machines. Both results are acceptable to the proposed cases.

6 CONCLUSIONS

We have analyzed how to introduce EDA in a core banking EA that allows programmers to create software bank applications quickly, efficiently and proving high performance. This evolution towards EDA allows the obtaining and exploiting of banking business events but it has associated challenges. We have analyzed them and proposed a set of solutions, which are: a business event definition based on an XML structure with our own semantic, a reference architecture to integrate EDA that identifies the specific integration points with the banking EA, and a definition of an event lifecycle that allows the incorporation and use of business events without interfering with the existing EA and its related governance processes. These solutions have been successfully validated in a proof-of-concept test bed that uses open source tools. Also, we have carried out non-functional validations of the selected tools, focusing on a performance evaluation.

Since our first results have demonstrated the workability of our approach, the future points towards further analysing the governance of an EDA. The core banking EA requires solutions that allow the cataloguing and managing of events to optimize their production, reuse and consumption. The monitoring of events and their lifecycle will also ensure the consistency of the EDA solution.

ACKNOWLEDGEMENTS

This work is part of the Center for Open Middleware (COM), a joint technology center created by Universidad Politécnica de Madrid, Banco Santander and its technological divisions ISBAN and PRODUBAN.

REFERENCES

Aihkisalo, T., 2011. A Performance Comparison of Web Service Object Marshalling and Unmarshalling Solutions. In *SERVICES 2011, 2011 IEEE World congress on Services.* IEEE.

Becker, J., Matzner, M., Müller, O., Walter, M., 2012. *A Review of Event Formats as Enablers of Event-Driven BPM*, Lecture Notes in Business Information Processing vol. 99, 2012, pp. 433-445.

Clark, T., Barn, B.S., 2012. A Common Basis for modelling Service-Oriented and Event-Driven Architecture. In *ISEC '12, Proceedings of the 5th India Software Engineering Conference,* pp. 23-32. ACM

Portal: ACM Library.
EsperTech 2013, *Event Stream Intelligence.* Available from: <http://esper.codehaus.org/>. (15 October 2013).
Etzion, O., Niblett, P., 2010. *Event Processing in Action*, Manning Publications. Greenwich, CT, USA, 1st edition.
Eugster, P., Felver, P., Guerraoui, R., Kermarrec,A., 2003. The many faces of publish/subscribe, *Journal ACM Computing Surveys (CSUR),* vol.35, no.2, pp.114-131. ACM Portal: ACM Library.
Maeda, K., 2012. Performance evaluation of object serialization libraries in XML, JSON and binary formats. In *DICTAP 2012, 2st International Conference on Digital Information and Communication Technology and it's Applications*, pp. 177-182. IEEE.
Malekzadeh, B., 2010. *Event-Driven Architecture and SOA in collaboration - A study of how Event-Driven Architecture (EDA) interacts and functions within Service-Oriented Architecture (SOA).* Department of Applied Information Technology 2010:056. Available from: University of GothenBurg. (15 October 2013).
Michelson, B. M., 2006. *Event-Driven Architecture Overview – Event-Driven SOA Is Just Part of the EDA Story.* Available from: Patricia Seybold Group.
RabbitMQ 2013, *RabbitMQ Messaging that just works*. Available from: <http://www.rabbitmq.com/>. (15 October 2013).
Taylor, H., Yochem, A., Phillips, L., Martinez, F., 2009. *Event-Driven Architecture: How SOA Enables the Real Time Enterprise* (1st ed.). Addison-Wesley Professional.
Vidačković, K., Kellner, I., Donald, J., 2010. Business-oriented development methodology for complex event processing: demonstration of an integrated approach for process monitoring. In *DEBS '10, Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems,* pp. 111-112. ACM Portal: ACM Library.
Weigand, H., 2011. The pragmatics of event-driven business processes. In *I-Semantics '11, Proceedings of the 7th International Conference on Semantic Systems*, pp. 211-218. ACM Portal: ACM Library.