"AN USE CASE ABOUT EVENT-BASED SYSTEMS APPLIED TO SERVICES-ORIENTED ARCHITECTURE AT SERASA EXPERIAN"

Carlos Neves Júnior
(Instituto de Computação - Universidade Estadual de Campinas, São Paulo, Brazil)
cneves@ic.unicamp.br

Hans Kurt Edmund Liesenberg
(Instituto de Computação - Universidade Estadual de Campinas, São Paulo, Brazil)
hans@unicamp.br

ABSTRACT

This article describes the experience and deliverables of a research supported and sponsored by Serasa Experian as part of its program for incentive of academic research. The aim of this project was to introduce the event driven architecture story to the company and how it extends the service-oriented architecture. In order to achieve this main objective, a real use case of an event-based system based on corporate scenarios at Serasa Experian was modeled and implemented using an integrated development environment (IDE) that is part of this research. This IDE based on Eclipse offers support for development of distributed event-based systems and has visual editors for modeling and specification of these systems. The use case was also deployed on a controlled runtime environment also described in this article, so its execution and the deployment effort could be simulated and tested. This paper presents the contributions of this experience of implementing a real corporate event flow and how this opportunity influenced this work.

Keywords

event-based systems, service oriented architecture, software engineering, behavioral modeling, reactive components, distributed computing

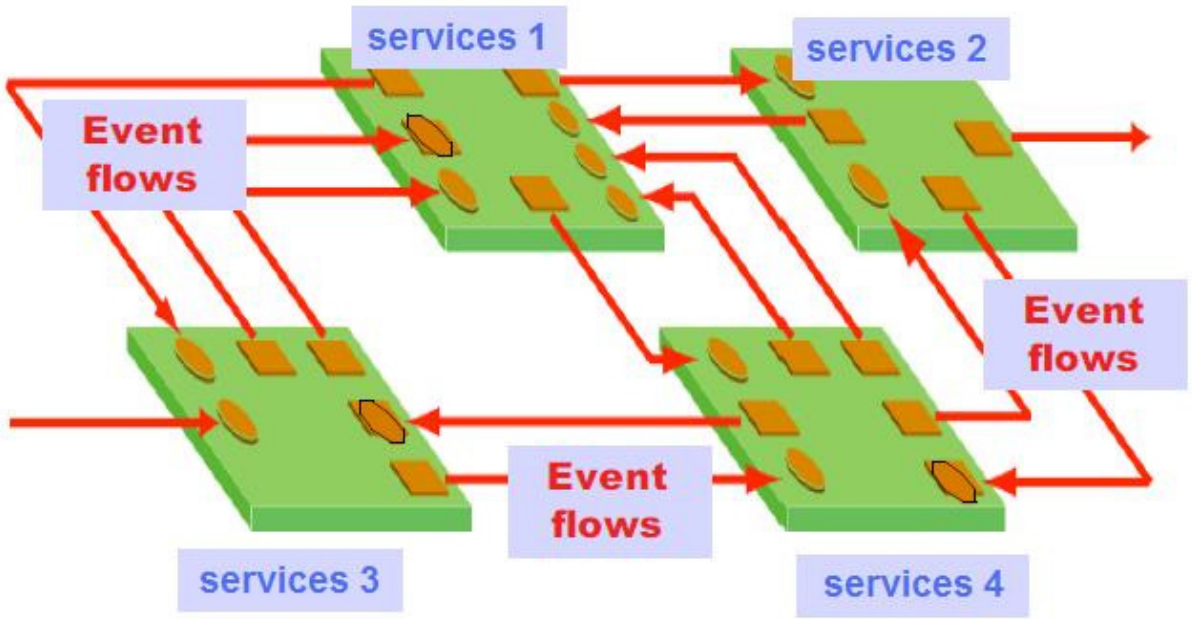
Acknowledgments:

The authors would like to thank Serasa Experian and all of its associates who contributed to this study and research. We are especially grateful to Djalma Padovani and Luciana Portela for their collaboration during the regular meetings and for providing relevant information for the completion of this work.

1. Introduction

The Service-Oriented Architecture (SOA) is an IT architectural strategy for infrastructure and business solutions that is well known and widely adopted by enterprises since the beginning of this century. The benefits of adopting SOA were easily assimilated by IT organizations and opinion-makers and it became an essential architectural pattern in the list of best practices for development of software to the enterprise. SOA allows reuse of software in a very effective way reducing the time for deployment and maintenance of applications what is very important due to the dynamic nature of the enterprise systems. A service is a defined functionality and/or resource that can be requested by one client. In SOA, the coupling between client and services is weak since the client does not need to know any information about the services implementation and rely only in a well-defined interface. Loose coupling between client and services is one of the key reasons why adoption of SOA simplifies management of the application and reduces dependency between system components. Many software vendors delivered products based on SOA to attend this market fever that definitively came to stay.

Due to the maturity of this topic, roadmaps of SOA are being discussed and some leading SOA and integration vendors are coming with their visions and strategies for the next steps. Some analysts consider we are entering in a second phase of SOA and it is not difficult to see what they called the event-driven SOA story. Event-driven SOA[15], also called SOA 2.0, is the interaction between events and services where the occurrence of an event can result in the invocation of one or more services. Event is an internal or external relevant thing that happens in the context of the business. Figure 1 represents a typical enterprise system where events can invoke services.



[Figure 1] Event-driven SOA – occurrence of events can trigger services

Event-driven SOA is part of a major architecture pattern called Event-Driven Architecture (EDA). EDA is an architectural approach to model information systems from a business event perspective. While pure SOA puts services at the center of the model, EDA is centered on business events. SOA tends to result in a synchronous communication style implementing the request/reply interaction model and EDA tends to be an asynchronous communication style implementing the event-based interaction model.

EDA can extend and complement SOA adding the connection between event and services and proactively invoking the functionality. The event-based interaction model present on EDA is implemented by publish/subscribe interface. This publish/subscribe event-based approach reduces even more the coupling between system components (client and service) and it makes EDA more adequate than pure SOA for some systems that require more independency of the components. Typically, in a heterogeneous enterprise system where processes are on cross functional organizations, it is important to reduce coupling to allow system evolution.

These are some of the concepts that motivated Serasa Experian to select our project in order to understand more the application of EDA on service-oriented platforms. Serasa Experian, a Latin American leading company of the financial and credit market, has a program [23] to incentive academic research and sponsored our project during nine months. They understand that event-driven SOA would be a possible step on the evolution of their infrastructure, so before the adoption of this trend and the decision to move with one specific leading software vendor in the area, this project contributed to bring the knowledge and set expectations about the benefits and requirements of such architecture pattern deployment in their environment. On the other hand, the engagement and support provided by Serasa Experian to this research certainly allow better quality and point different perspectives to this work. As per the official schedule of this project, it was aligned between the parts that one use case based on a real event-based system of the company would be modeled and implemented using an integrated development environment (IDE). This IDE, a development tool for specification and coding of distributed event-based systems, is also a deliverable of the project and its functionality captures the requirements of this research. The IDE is developed as an Eclipse[8] plug-in and so it allows the use of the common resources of the Eclipse environment and extension of the IDE.

This article is organized in the following way: Section 2 presents some fundamental concepts of event-driven architecture that will be used in this document. An informal description of the event-based use case of a real scenario at Serasa Experian is the content of the Section 3 of the document. On Section 4, it is presented the overall implementation aspects of this project, the implementation of the IDE used to specify and code the use case and also more details about the implementation of the use case itself. The use case implementation is covered from its specification to its deployment and execution with considerations about the project decisions. This paper finishes with conclusions and recommendations for future work in Section 5.

2. Background

In this section, it is briefly provided some background information on the event-driven architecture and the event tooling. More details on the event-driven architecture can be found in the cited references.

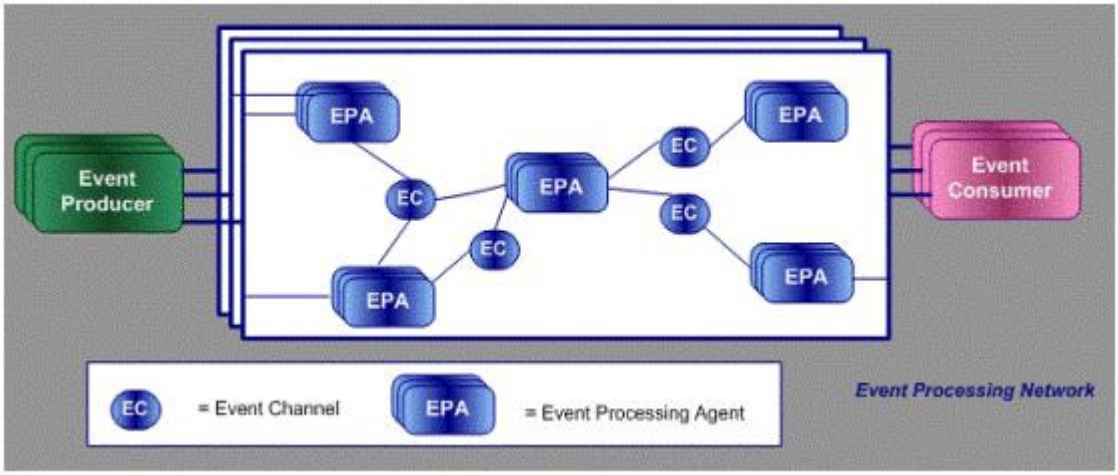
2.1. Event-Driven Architecture

Event-driven architecture [15] is an architecture pattern used to design systems with extreme loosely coupled components that may produce, consume or process events. The components of these systems are typically highly distributed. These systems designed using EDA principles are classified as distributed event-based systems[16] (DEBS) or simply event-based systems.

The following classification for the components and terms encountered on an event-based system will be used throughout this document:

- An event **producer** is a component that only generates or produces events. It has no control and information of which other components will receive and how they will process the generated event. The producer detects raw events and format event information in order to publish in the event system.
- An event **consumer** is on the other hand a component that only consumes events. It has no information about how the received event was generated and which other components generate it. The consumer subscribes for particular event information and may take actions when events arrive.
- An **event processing agent or EPA** is a component responsible for the event processing activities. It has the ability of consuming and producing events as well perform event processing based on specific rules. As a result of this processing, new events can be generated and publish in the event system. Typically there is one event pattern language to describe EPA's behavior.
- An event **channel** can also be considered a component responsible for transporting and delivering of events. An event channel can implement different policies like for example ensure first in first out delivery of events and enforce message ordering.
- An **event processing network or EPN** is a collection of EPA components organized to determine a network of communicating lightweight processing engines. EPN is a strategy for optimizing the use of engines and for breaking down complex event processing problems into simple steps.

Figure 2 illustrates the high level view of the components of an event-based system and includes an event processing network.



[Figure 2] Components of an event-based system

2.2. Event Tooling

In order to specify one distributed event-based system, i.e. define the list of events, the event processing rules and the event processing flow for a certain problem, it is important to have access to event development tools. An appropriate event development tool would contain editors that implement languages for describing the event flow and organization of the components and editors that implement languages for describing event patterns (rules).

Event tools are also important to manage the runtime of an event-based system, so features like control of subscriptions, monitoring and administration of the event system infrastructure, statistics of the event flow, visibility and transport of the event generation are important functionality in order to support the lifecycle of an event-based system.

Some leading software vendors like IBM and Oracle [1] offers on their SOA-based products some sort of event development tooling to support implementation of event-driven architecture. An event tool developed as part of this project is used to specify an use case based on a Serasa Experian real scenario. This same event development tool support coding, deployment and execution of the event-based system.

3. Use Case – mapping the real world

During the definition of this project in conjunction with its sponsor, Serasa Experian, it was decided that an event-based use case based on a real scenario would be tentatively mapped by the project and simulated for analysis. During the regular project meetings, the team revised some real IT and business scenarios and the agreement was on the mapping of the event flows originated from their customer service e-mail system. A notable amount of e-mails are sent from and to Serasa

Experian's customers, so they want to monitor this service, proposing an additional control of this system. It would give a chance for improvement of the communication with their customers and also keep their internal infrastructure monitored for quick problem resolution. In the sequence, there is an informal description of this use case.

Basically, it was mapped three e-mail flows that exist in the current IT and business systems. These three notable mailboxes listed below are the resources to be monitored:

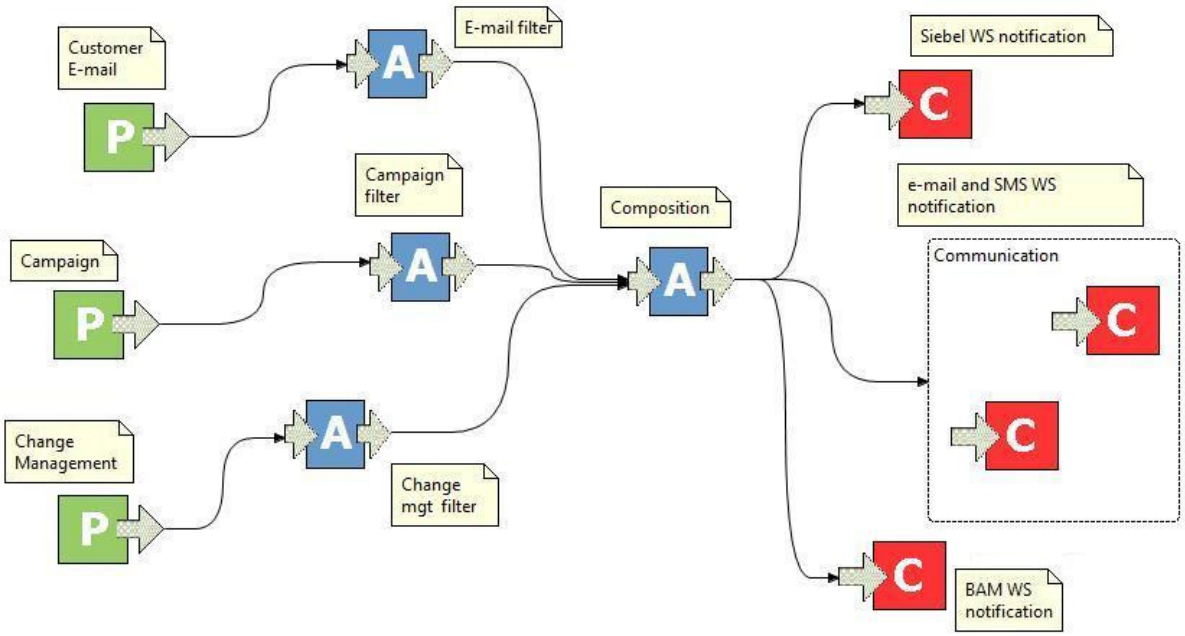
- **Customer E-mail inbox**: This e-mail inbox receives all e-mails sent from external customers. Customers typically report a problem or complaint, ask for help or submit a form. The e-mail management is done manually and there are human agents and coordinators checking the arrival of e-mails.
- **Campaign outbox**: This e-mail outbox reflects all e-mails sent out for customers due to an active market campaign. A market campaign would announce a new service or simply keep communication with the customers. A campaign is automatically generated by one automated tool that reads a list of customer e-mail addresses (campaign list).
- **Change management inbox**: This e-mail inbox is for internal IT activities and is used to register all changes in the IT system. Any change of the system will be notified on this mailbox. IT Administrator creates those changes notifications.

This use case consists of monitoring these three mailboxes and generates events for each occurrence of e-mail. Each mailbox will be monitored by one **producer** that will be responsible to generate these events. It means we have three producers in the specification as observed in figure 3. The producer co-located with the Customer E-mail inbox will generate an *ordinary e-mail event* for each e-mail that arrives in the inbox. Some special events can and are also created by this producer: an event marked as *unread count event* is generated if a certain number of unread e-mails are left in the mailbox and this count reaches a parameterized threshold (default limit is 10). The same occur if a certain number of e-mails marked as urgent arrives in the mailbox and a parameterized threshold is reached (default limit for urgent e-mails is 5), an *urgent count event* is generated. So those events are generated by the producer and sent to the channel that connects this producer to an **EPA** responsible for filtering events or generate new events (E-mail filter EPA). This use case specification can be followed in the figure 3 since it presents the components of the event-based system.

Similar role is performed by one producer that is co-located with the Campaign outbox. This producer generates a *campaign event* to indicate a certain campaign was initiated. Associated to the campaign event, we have a list of customer e-mail addresses that will be notified as part of the campaign. Note that an event when is generated by one producer can also carry some data for processing in the event system and in this case the campaign event carries the event list with all customer e-mail addresses.

A change in an IT or business application is identified by one e–mail that arrives in the mailbox for Change management. The producer co-located with it generates one *change event* for each e-mail that arrives in that inbox representing a change occurred.

All producers are connected to EPA components that process events and apply some rules in order to filter, forward or generate new events. An EPA also can execute some actions as result of their processing.

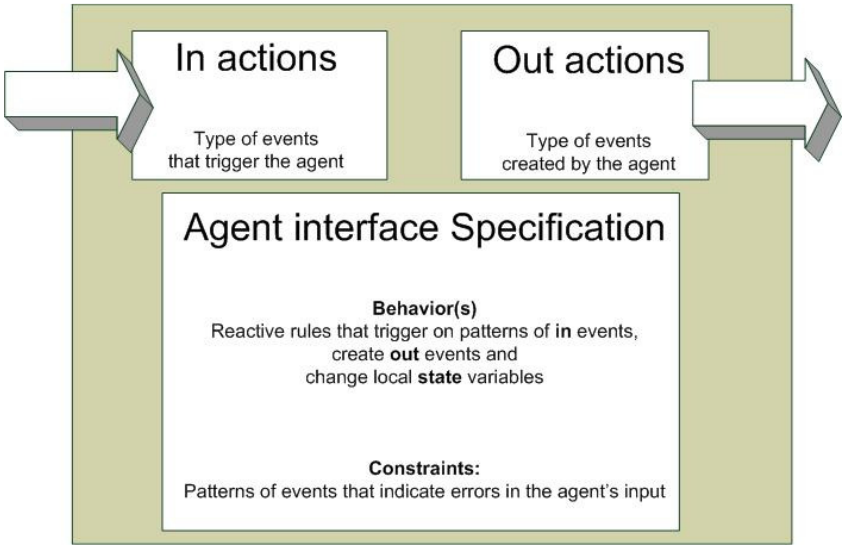


[Figure 3] EPN representing event flow of Use case about e-mail management (P=Producer, A=event processing Agent, C=Consumer)

The idea is that those event notifications would flow through the EPN of components and depending on the reactive rules of each EPA specification, some events could finally be directed to the **consumers** accordingly to the event flow presented in figure 3.

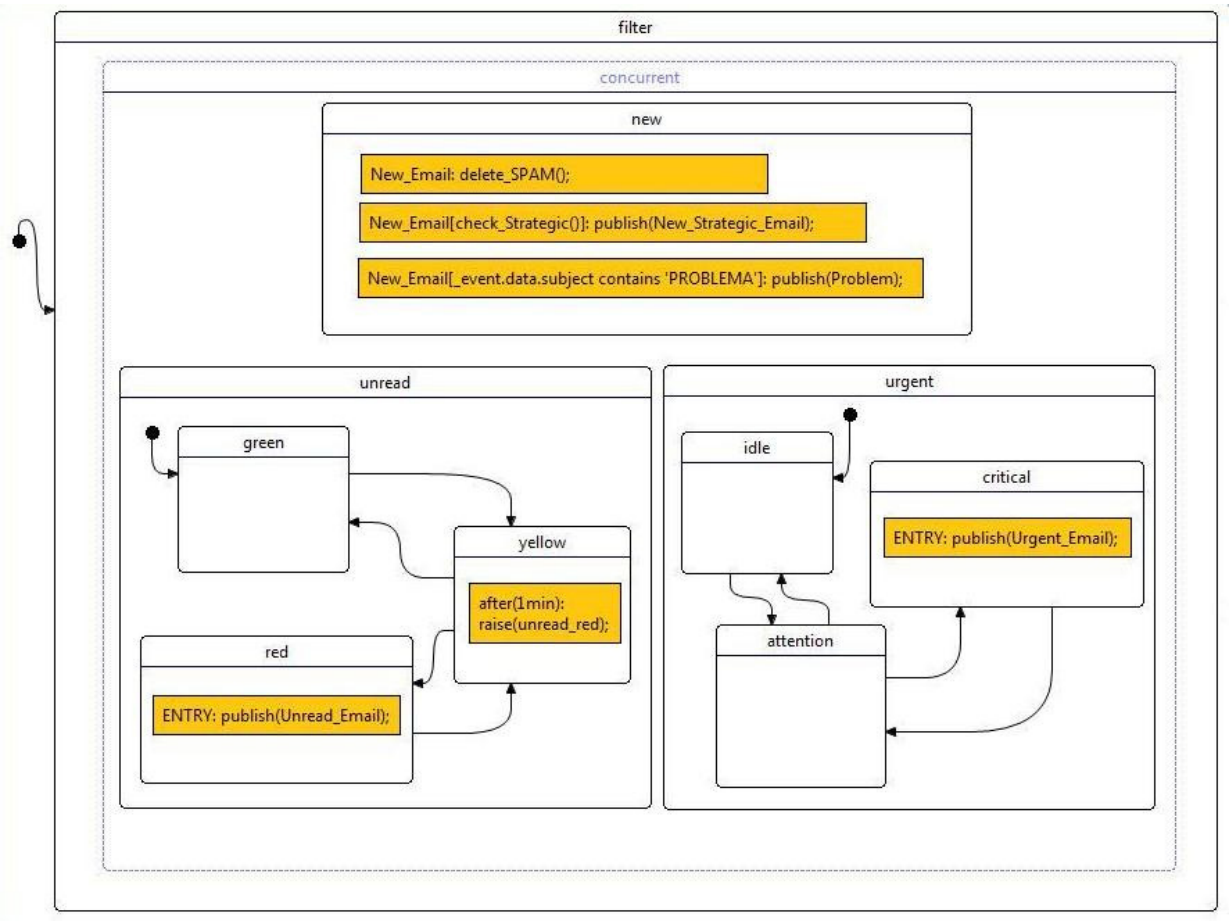
The consumers then receiving those events could react invoking some functionality and services. In this specification, there are 4 consumers that are responsible to notify internal systems and humans (Siebel, BAM, SMS and e-mail) depending on the arrival of the events and this notification is done via web services. This is the connection of events and services that is promoting the event-driven SOA in this use case.

Figure 4 shows the internal organization of each EPA and its interfaces. Note that each EPA has reactive rules that process events. In this project, EPA's reactive rules or event patterns are specified using *Statecharts[12]*, a visual language for behavior specification of reactive components. Some extensions to the standard *Statecharts* specification were proposed as part of this work and they will be described in the next section.



[Figure 4] interface of an event processing agent (EPA)

The figure 5 shows part of the *Statecharts* specification that describes the behavior of the EPA filter associated with the Customer E-mail producer.



[Figure 5] E-mail filter EPA – event pattern/reactive rules using *Statecharts*

In order to understand how an EPA processes events, follow figure 5 description. This E-mail filter *Statecharts-based specification* has three concurrent states for processing of the events *new e-mail*, *urgent e-mail count* and *unread e-mail count* forwarded by producer. At each *new e-mail* notification, some rules are applied like the one that initiates an action to discard SPAM. Note that other two rules dependent on the arrival of *new e-mail* event can eventually generate (publish action) other new events if their conditions are satisfied. These new events are generated when a strategic customer sent an e-mail and/or an e-mail reporting problems is received.

The state *unread* evolve to a red state if the *unread email count* persists high and then forward the unread e-mail notification ahead (publish action) in the EPN. A similar situation occurs with the state *urgent* that can forward the notification for urgent e-mail if this e-mail count stays high and its child state *critical* is then activated.

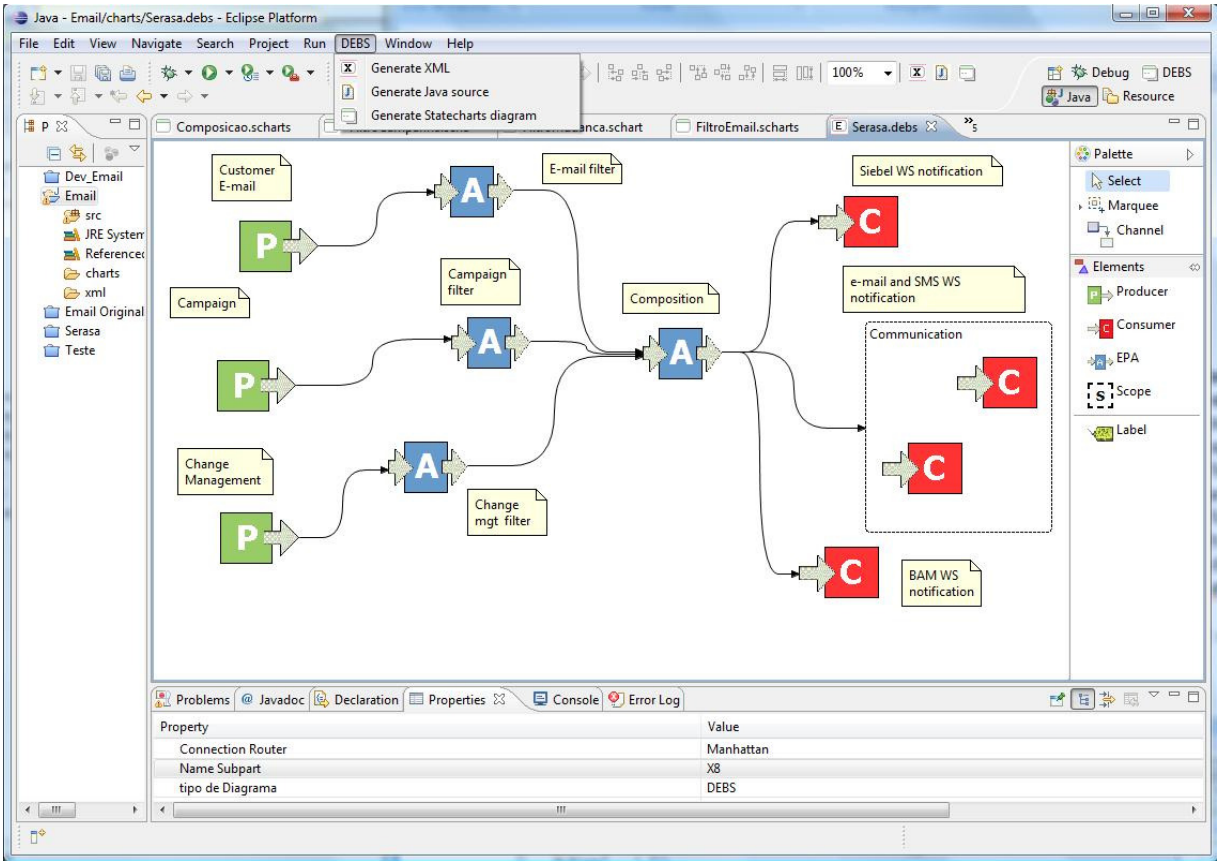
The use case was modeled and developed using an Eclipse plug-in for DEBS. The implementation of this plug-in and use case is discussed in the next section.

4. Implementation and Results

This section introduces the implementation and deployment aspects of some of the project deliverables. These are key results of our research.

4.1. The IDE for Distributed Event-Based systems

An integrated development environment (IDE) for development of distributed event-based systems is a deliverable of this work. Its development is an updated version of similar prototype of IDE functionality and *Statecharts-based* Editor from previous works [18][13]. This IDE was developed as a plug-in of the Eclipse framework [8] and all components are coded in Java. This IDE offers support for development of the individual building blocks of a distributed event-based system (producer, consumer, event processing agent and channels) and as well support for design of the whole distributed view of a DEBS associating each component in the form of a network of components which consists on event processing network with producers and consumers. This IDE and the project itself are currently called *Xchart* and would be available for download at Eclipse labs *[9]*.



[Figure 7] Eclipse plug-in for distributed event-based system development

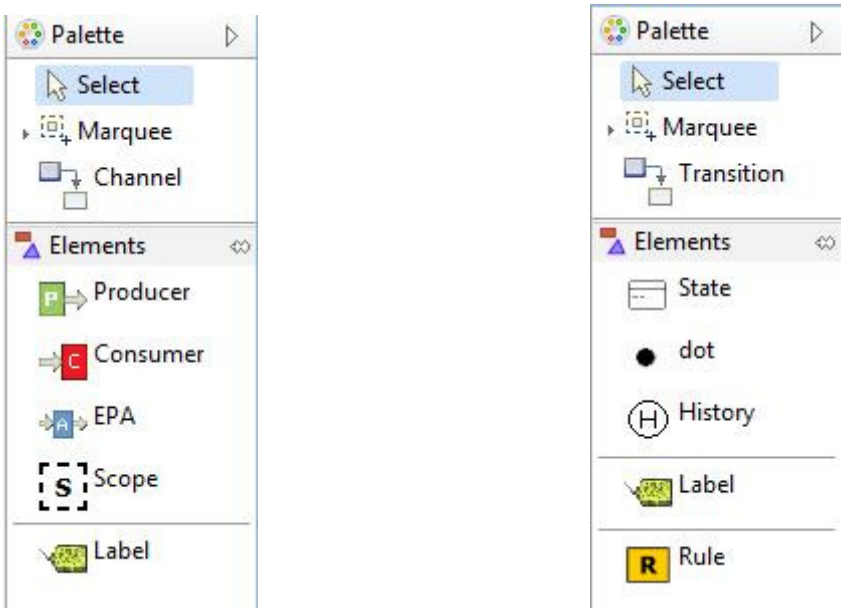
Figure 7 illustrates the look and feel of the Eclipse-based IDE for development of DEBS. The Eclipse-based IDE has two main visual editors:

- **EPN editor**: one editor for specification of the whole distributed view of event-based systems. This IDE's editor offers controls for design of an EPN (collection of EPAs) with producers and consumers. This editor allows automatic generation of XML content that stores the distributed view and associations of each component and since no standard exists on that area, it is used a specific format (*DEBSXML*). This editor also automatically generates Java source code for producer, consumer and EPA components so developers only need to customize the specific domain of the problem for event generation (producers' code) and event-driven actions (consumers' code) with appropriate Java code.

- **Statecharts editor**: one editor for development of the individual event processing agent components using an extension of *Statecharts* notation. For each EPA component, it is associated one *Statecharts* diagram that describes the behavior of its reactive rules like in [2]. This editor has option

that allows automatic generation of *SCXML* content from the *Statecharts* visual specification.

Figure 8 presents the specific palette options that are available for each editor:



[Figure 8] Eclipse palette options for EPN and *Statecharts* editors.

It is important to register that the *Statecharts* variant implemented in the Eclipse plug-in was extended as part of this work in order to give the adequate capabilities and also simplify the design and code of the use case informally described in the previous section. For instance, the <rule> tag is not officially available in the *SCXML* [22] as a standard tag, but this capability was introduced. A rule behaves like a transition in terms of executable code, but it does not deactivate or active states. The rule only executes actions and a similar concept is also presented in a previous work that describes the xchart language [19]. The following piece of code automatically generated by the IDE's *Statecharts* editor shows how XML content will look like for the <rule> tag.

```
<state id="new">
	<rule event="New_Email">
		<invoke type="java" src="delete_SPAM"/>
	</rule>
	<rule event="New_Email" cond="check_Strategic()">
		<send target="external" event="New_Strategic_Email"/>
	</rule>
	<rule event="New_Email" cond="_event.data.subject contains 'PROBLEMA'">
		<send target="external" event="Problem"/>
	</rule>
</state>
```

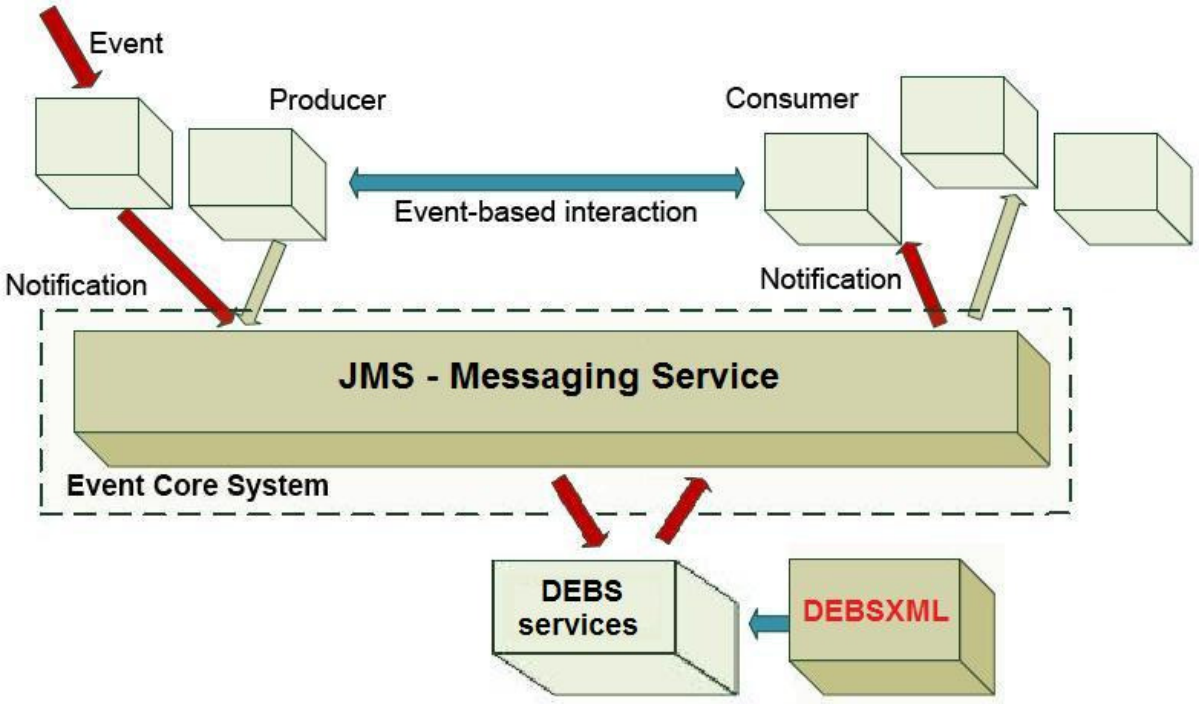
4.2. Use Case Coding and Deployment

The coding of any use case should be facilitated by the automatic generation of Java source code provided by the IDE's EPN editor option. All generic APIs of producers, consumers and event processing agents like the publish/subscribe interface should be automatically available for the developer that will only focus the specific domain of the problem and will need to code only the event detection and actions.

For this particular use case of the project, the story was a little bit different since the IDE was developed as part of the project and so the first version of the producer, consumer and EPA components APIs needed to be done from the scratch. It means this project gave a notable contribution for future implementations being the first real validation.

All development (use case and IDE) was done using Java and Eclipse[8][20]. The components of an event-based system must interact via publish/subscribe mechanism. The event system middleware is implemented using JMS (Java Message Service [21]) as the messaging notification service and additional custom Java objects (POJO) for other DEBS services. These DEBS services are responsible for dealing with some specification data like the scopes definition (grouping) and all subscriptions that are derived from the *DEBSXML* specification automatically generated by the EPN editor.

Figure 8 illustrates the components of the system on the runtime environment.



[Figure 8] Runtime – components of the DEBS execution

There is one state machine engine derived from the Commons SCXML project [4] that is used for runtime of the *SCXML* content that is generated by the IDE in an automated way for each *Statecharts* diagram. It was necessary to custom and extend its source code since the default functionality provided by the Commons SCXML engine was not enough like for example the support added for rules in *Statecharts*. Another two aspects that motivated the extension of this engine is the need to add support for temporal events that were not well addressed and are very important for DEBS specification and also the specific platform customization of the <invoke> tag to implement invocation of Java methods as well Java functions embedded in the SCXML code. The SCXML engine is definitively one component that demanded changes and deviates from the original Commons SCXML source code. The extensibility of the tool is one of its goals.

In terms of the deployment, once the use case coding was completed and verified on the desktop, all software was manually copied to a single server for execution. A basic Windows 2003 SP2 server with single Xeon CPU 2.33 Ghz and 2 Gb RAM was dedicated for execution. The manual preparation of this server required installation of the JMS OpenMQ4.5 and creation of all component connections to JMS. Each component and service of the environment is started manually by administrator of the system. An automated method to start up the system and all of its components is a kind of suggestion for future work.

The deployment and activation of the application was not done at the Serasa Experian IT environment. In order to have flexibility with our deadlines, the agreement was to simulate the application on external servers and monitor external mailboxes since an approval to deliver the solution internally at Serasa Experian could introduce some delays. Some of Serasa Experian's associates generate little traffic on this external server and mailboxes following by their experience on the common customers requests. This data is currently being used to generate additional traffic since it is a good representation of typical customer interaction and some statistics and comparisons about performance will be subject of future reports. In the current research, the correctness of the rules and event processing was the main objective.

5. Conclusions and Recommendations

One of the gaps of previous cycles of this research was really the absence of a real example to be a reference for our work. This is by far the most important achievement of the project, the possibility and opportunity to implement a solution that is closer to the real world requirements and that enforced a solution for real problems. It is important that new real solutions should be modeled since the author recognizes it is difficult to design those systems. During the design phase of this use case, the author needed to create multiple versions of the design until the final one was implemented. It means that this old recommendation is still applicable: more

experimentation of distributed event-based systems is recommended since it is expected that a number of specific components, consumers and producers, will be largely available and the plan is to study a better way of re-using this software on future implementations. We have the concept of connectors (specialized producers) and providers (specialized consumers), so one idea would be to package these solutions and recommend a catalog of reactive components (patterns [6][10]) for certain domains. The author's impression is that even with the use of visual languages, the modeling and design task of the real use case was complicated and more experience and patterns should be collected for analysis. The review of other methods for design is also part of that analysis [3][11][7].

The core of this research is about event driven architecture and event based systems. The connection to Services oriented architecture is important, but the message that this research tried to bring for Serasa Experian is that EDA is a wider story and they should avoid short-sided event architecture implementations. This research served to orient the Serasa Experian to not miss out on opportunities for information flow processing and analysis taking advantage of events.

The author wants to make a special recognition to the Serasa Experian team for their great contribution on this project that beyond the sponsorship of this research, provided feedback on top of the features. These are the areas as per their feedback that the team might need to work in future versions of this project: improve event tooling for monitoring and audit of events during runtime and prepare a friendly management of the event specification catalog.

In the author´s point of view, CEP (complex event processing [14][5]) is another area that also would dedicate a better view of our work.

6. References

[1] Alves, Alexandre. *A General Extension System for Event Processing Languages*, The 5th ACM International Conference on Distributed Event-Based Systems, July 2011.

[2] Blanco, Rolando and Wang, Jun and Alencar, Paulo. *A Metamodel for Distributed Event Based Systems*, The 2nd ACM International Conference on Distributed Event-Based Systems, July 2008.

[3] Booch, Rumbaugh, and Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1998.

[4] Commons SCXML, Apache foundation, http://commons.apache.org/scxml/ , 2012.

[5] Cugola, Gianpaolo and Margara, Alessandro, *Processing Flows of Information: from data streams to complex event processing,* The 5th ACM International Conference on Distributed Event-Based Systems, July 2011.

[6] Deepak Alur, John Crupi and Dan Malks. *Core J2EE Patterns: Best Practices and Design Strategies*. Second Edition. Prentice Hall PTR, Upper Saddle River, N.J., 2004.

[7] Drusinsky, Doron. *Modeling and Verification Using UML Statecharts*, Elsevier, 2006.

[8] Eclipse Foundation, *The Eclipse project*, http://www.eclipse.org, 2011.

[9] Eclipse labs, xchart, Visual Editor for behavioral modeling and distributed event-based systems, http://code.google.com/a/eclipselabs.org/p/xchart/ , 2012.

[10] Fowler, Martin. *Patterns of Enterprise Application Architecture*, The Addison-Wesley, 2003.

[11] Gomma, Hassan. *Designing Concurrent, Distributed, and Real-Time Applications with UML*. Addison-Wesley, 2000.

[12] Harel, David. Statecharts: *A Visual Formalism for Complex Systems*. *Science of Computer Programming*, 8(3): 231-274, June 1987.

[13] Kollross, Diogo and Liesenberg, Hans Kurt Edmund, *Sins: Um Editor Xchart na forma de Plugin para o Ambiente Eclipse*. MSc's dissertation, IC/UNICAMP, Campinas/SP, October 2007.

[14] Luckham, David. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems,* Addison Wesley Professional. May 2002.

[15] Michelson, Brenda M., *Event-Driven Architecture Overview: Event-Driven SOA is Just Part of the EDA Story*, http://dx.doi.org/10.1571/bda2-2-06cc, 2012.

[16] Mühl, Gero and Fiege, Ludger and Pietzuch, Peter. *Distributed Event-Based Systems,* Springer, 2006.

[17] Neves Júnior, Carlos, *Gerente de Distribuição do Ambiente Xchart em JEE*. MSc's dissertation, IC/UNICAMP, September 2005.

[18] Neves Júnior, Carlos e Liesenberg, Hans Kurt Edmund. *A Visual Programming Approach for Development of Event-Based Systems*. 7o. CONTESCI, São Paulo, Brasil, May 2010.

[19] Nogueira de Lucena, Fábio. Xchart: *Um Modelo de Especificação e Implementação de Gerenciadores de Diálogo*. PhD's thesis, IC/UNICAMP, Campinas/SP, December 1997.

[20] Steinberg, Dave and Budisnky, Frank and Paternostro, Marcelo et al, *EMF: Eclipse Modeling Framework (2nd Edition)*, The Addison-Wesley, 2009.

[21] Sun Microsystems Inc. *Java Message Service Specification, Version 1.1*. http://java.sun.com/products/jms, 2010.

[22] W.W.W.C. (W3C), *State Chart XML (SCXML): State Machine Notation for Control Abstraction*, W3C working draft April 26th 2011, http://www.w3.org/TR/scxml/, 2012.

[23] Programa de Incentivo a Pesquisa Aplicada Serasa Experian & UNICAMP, http://www.serasaexperian.com.br/pesquisaaplicada/, 2012