Baggage Tracking System

André Albuquerque José Alves Marques

Instituto Superior Técnico

Av. Rovisco Pais, 1

1049-001 Lisboa, Portugal

andre.m.de.albuquerque@ist.utl.pt; jose.marques@link.pt

ABSTRACT

The loss or mishandling of luggage in airports is increasing nowadays, tremendously raising its associated costs. The presented paper proposes, within the scope of a project done by Link Consulting, a system capable of improving the whole baggage handling process, by monitoring the processed luggage in real-time. It's expected that the constant monitoring detects possible errors in a timely manner, allowing a proactive attitude when correcting this kind of situations. This will increase the performance of the currently used baggage handling systems and therefore will reduce their associated costs.

A totally functional prototype that implements the proposed architecture was developed: it processes the data that results from the luggage detection; it presents real-time information reports; it stores permanently the information that results from luggage monitoring and it also allows the access to historic tracking information using a Web application. This system increases the monitoring detail when compared to current monitoring systems because it allows the individual tracking of each processed bag, since the information used is obtained by reading the tag placed in each bag. Therefore, system granularity is increased and it's possible to detect exceptional situations happening only to a single bag. This reduces the reaction latency on this kind of situations, allowing smaller correction times that avoid passengers being early informed about the mishandling of their luggage.

Keywords

Baggage tracking system, baggage handling process, real-time event processing, RFID technology, flexible baggage monitoring.

1. INTRODUCTION

The baggage handling process in the airports it's directly related with complexity and uncertainty factors in most of the passenger's opinion, as many other aspects related with the air traffic. Many passengers feel some uncertainty when they see their luggage disappearing behind the check-in curtains.

The luggage that for some reason is lost during the process, despite being a small percentage, continues to be one of the main concerns to air passengers. The increase in passengers' traffic and the recent effort in strengthening the security measures increased the global problem, implying an annual cost of 3.8 billion USD to the air transport industry [1]. The uncertainty shown by passengers towards the baggage handling process and its inherent errors is also constantly increasing. Both situations could be mitigated if there was a system that allowed:

- Passengers to follow the path done by their luggage by using a Web application or by subscribing a service that allowed them to receive relevant information about the current state of their luggage in different points of the handling process (e.g. when the baggage enters the plain, the passenger receives an SMS informing him about its luggage state);
- Baggage handling personnel to know in each moment the path's segment where each processed bag is. With this information, if the luggage wasn't received in the expected place, it could be spotted and therefore the possible mistake could be corrected in a timely manner.

If, for some reason, someone need to find a specific baggage to be rerouted in the handling process (e.g. because the passenger didn't embark in the expected flight) it's very important to have mechanisms that allow the immediate baggage tracking with enough accuracy to define a small search area within the airport, enhancing the search process. Therefore, the main objective of this work is to propose a solution for the cited problem, in the form of a tracking system capable of presenting process' performance indicators and allowing the airport personnel to know each baggage location in any handling process' segment, from the entrance of the check-in to the entrance of any flight.

It will be possible, using the proposed system, to know in each moment the location of each bag, in which state the bag is, to which flight is associated, what the expected path is and which segments were already travelled, what alerts were raised concerning the bag or which bags embarked in some specific flight.

Using the information generated by constant monitoring the handling process, some process performance indicators will be supplied: the average time that a bag takes to travel a specific segment; the number of bags correctly loaded, not loaded, unloaded and lost; the deliver time of each flight (measured between the time when the first bag exists the plain and the arrival of the last bag in the deliver baggage carousel).

The presented article describes the handling process monitoring system here introduced and will exhibit the following structure:

- Section 2 introduces the airport baggage handling process and the main event processing concepts;
- Section 3 will show the proposed system's architecture, with emphasis given to the event processing component (EPC);
- The way how the system was tested and consequently simulated in different scenarios and the evaluation of the developed work are shown in Section 4;
- This article will be summarized in Section 5 along with the main conclusions obtained and the recommended future work.

2. RELATED WORK

To enable the easier comprehension of the baggage handling process monitoring problem, in this section will be shown some of the studied works that in some extent are related with the article's subject.

2.1 Baggage Handling Process

The baggage handling process is composed by a various steps, from the initial baggage check-in to the moment when the passenger picks it on the baggage carousel. In Fig. 2.1 each step of the baggage handling process is represented using a block diagram [2].

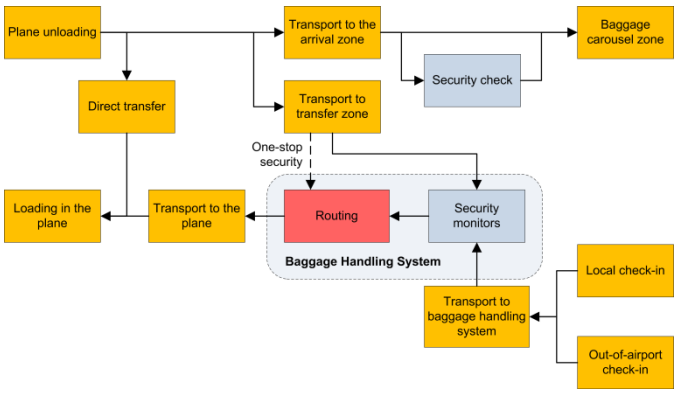

Fig. 2.1. Diagrama representativo de um processo de tratamento de bagagens [2].

The coordination between every airport system that participates in the handling process is only possible because of the messages exchanged between each system, allowing the baggage routing, the reconciliation between passengers and their luggage and other related services to be successfully automated. The *Baggage Information Messages* (BIM) are the messages exchanged between airport systems and are well defined by IATA in [3].

The *Baggage Source Message* (BSM) is one BIM type that defines the association between a bag and a flight and is used by the automated baggage routing systems to correctly route each bag to its respective flight. The monitoring system here presented uses the information encapsulated in each BSM to know which path will be monitored for each bag.

The monitoring system also uses the *License Plate Number* (LPN) defined by IATA in the Resolution 740 [3] to uniquely identify each passenger's bag in each trip. A LPN is a 10 digit number composed by 3 fields: the LPN type, the entity identification of the LPN emitter and the identification of the baggage. Each bag should have a unique LPN to allow the correct and unambiguous identification of every single bag.

2.2 Event Processing Concepts

The monitoring system here proposed uses event processing extensively so it will be here introduced the main concepts used in event processing.

In [4] *event* is defined as an occurrence in an external system, which is something that has happened, or is contemplated as having happened in that system or it is a programming entity that represents such an occurrence in a computing system. Applying this definition to the handling process context, the act of reading a baggage in a certain location (occurrence in an external system), the consequent XML message sent to the Event Processing Component (EPC) (representation of an occurrence that has happened) or the objects sent between EPC sub-components (programming entity that represents an occurrence) are defined as events.

Every Event-driven application is composed by entities that emit events (*Event Producers*); entities that receive events (*Event Consumers*); and entities responsible for processing the events (*Event Processors*) and if needed, sending new events to the event consumers after processing the received events. E*vent processing* means any computation that performs operations on events, including reading, creating, transforming and deleting events.

The operations realized by an event processor are defined in an *Event Processing Language*, or EPL. Actual EPLs can be rules based, script based, based on SQL extensions or based on other relational algebra languages. The system here presented uses the Oracle Complex Event Processing (CEP) tool to enable event processing. The EPL used by Oracle CEP uses an SQL extension developed in Stanford [5].

The event producers, consumers and processors should be linked between them to enable the processing of generated events that will thereafter be consumed, creating an *Event Processing Network*, or EPN. The Oracle CEP allows the inclusion of other elements besides the event producers, consumers and processors. As an example, there can be elements in the EPN that encapsulate Java code and therefore can contain business logic. In an Oracle CEP's EPN there can be the following elements [6]:

- Processors, configured using an EPL and responsible for processing normalized events arriving by an input channel and as a result, they could also generate new events to an output channel;
- Channels, responsible for linking elements in the EPN and for enabling the events flowing between EPN elements;
- Streams, with the same role as channels but allowing their content and performance to be monitored using Oracle CEP Visualizer tool;
- Adapters, responsible for allowing the interaction with the outside of the EPN; they have several interfaces available to enable the correct reading of events from many sources or to send events to many different receivers;
- Beans, that can be linked to a channel or a stream and are activated when they receive an event to be processed. Beans can encapsulate Java code and generally are POJOs.
- Event Beans, identical to the beans described above but as they implement Oracle CEP's API they can be managed and monitored by Oracle CEP tools;
- Caches, used as temporary storage for events. They can be used to enhance the application performance by storing events and/or objects frequently accessed.

2.3 Existing Solutions

Actually, there is some baggage tracking prototypes systems, namely the prototype implemented in the Beijing Airport, described in detail in [7, 8, 9]. In those works, Ting et al. define gradually a baggage tracking system architecture similar to the architecture proposed in this work that also uses RFID as identification method. The implemented prototype on Beijing Airport was based on a distributed application with the primary objective of tracking each baggage within the space airport.

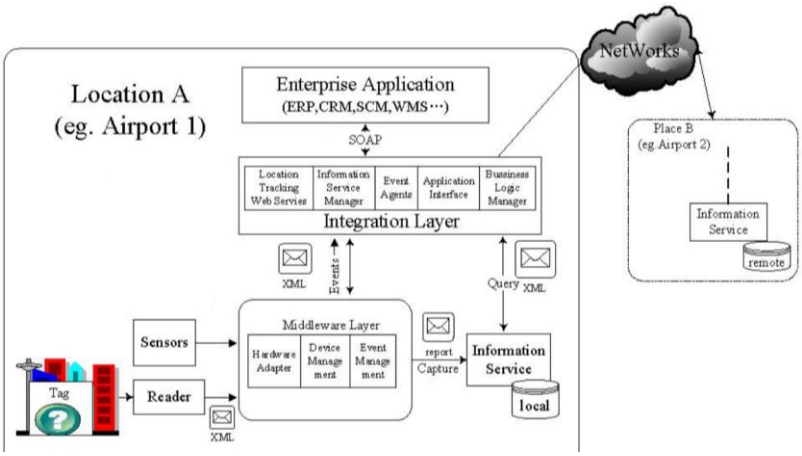

Fig. 2.2. Ting et al. proposed architecture**.**

Ting's tracking system was based on tagging each baggage with a RFID tag that was detected by the RFID readers displaced along the airport, producing location data about each baggage that was processed by the system to obtain intelligible information.

For efficiency reasons, the data obtained from the reads must filtered and agglomerated, creating new complex events after this initial simple event processing (each baggage read done by one reader can be considered a simple event). The location information of each bag can be obtained remotely using a Web Service invocation (cf. Fig. 2.2) and therefore the presented architecture can be considered a Service-Oriented Architecture (SOA) in some extent.

The data captured by the RFID readers is sent to the system's *Middleware* layer. The Middleware layer is responsible for the interaction with each RFID reader, for the management of the different readers connected to the system and for the event filtration and agglomeration, sending only relevant events to business applications or to the persistent storage. The concept of "logic reader" is also implemented in this layer, enabling the merge of several event streams from each physical reader in one unique event stream, represented by only one "logic reader" in the application side.

The *Information Service* (IS) contains a local repository to store relevant location information produced by the processing of the received raw events from the RFID readers. This component is also responsible for answering local queries, obtaining the needed information from the local repository.

The *Integration* layer allows other equivalent systems to query the local repository using a Web Service invocation. The messages exchanged between systems are XML based and the Web Service description, discovery, etc. uses WSDL and UDDI. If some query can't be answered locally, the *Information Service Manager* in the Integration layer is responsible for searching a system that can correctly answer to the query and to interact with it, obtaining the expected answer.

3. SOLUTION'S ARCHITECTURE

The system's architecture can be seen in the diagram on Fig. 3.1. The diagram shows us the main components of the proposed architecture: the *Event Processing Component* (EPC) responsible for the event processing; the *Business Indicators Component* (BIC) responsible for creating and presenting the process indicators; the *Event Repository* (ER) responsible for the persistent storage of events; the *Interface Application* (IA) responsible for allowing passenger and airport personnel to access information about the state of their luggage or the handling process respectively; the *Enterprise Service Bus* (ESB) responsible for the event routing and finally, the Event Simulator (ES) created to enable the simulation of real scenarios by sending event sequences to the system that will exercise the system's behavior.

The EPC receives the events and processes them according to the processing rules defined, gradually enriching the initial raw events that enters the system until complex events with relevant business information are sent to the BIC so that the presented *dashboards* can be refreshed with actual process information. The EPC receives events not only from the *Tags* layer but also events from the airport external systems. There can be events representing an association between a bag and a flight (coming from the Departure Control System, DCS) and there can also be events representing the arrival or departure of a specific flight (this type of events come from the airport's Operational Management System, OMS). The EPC is also responsible for storing its internal state and the received and generated events in the ER.

The BIC is composed by the interfaces that enable the event reception with the information needed by the dashboards, by the conversion mechanisms responsible for converting the received events in meaningful information understandable by the BIC that can be used later to refresh the reports shown to users and it's also composed by the report models that will be shown to users. The report models and the information used to fill the reports are stored temporarily in specific caches that belong to BIC.

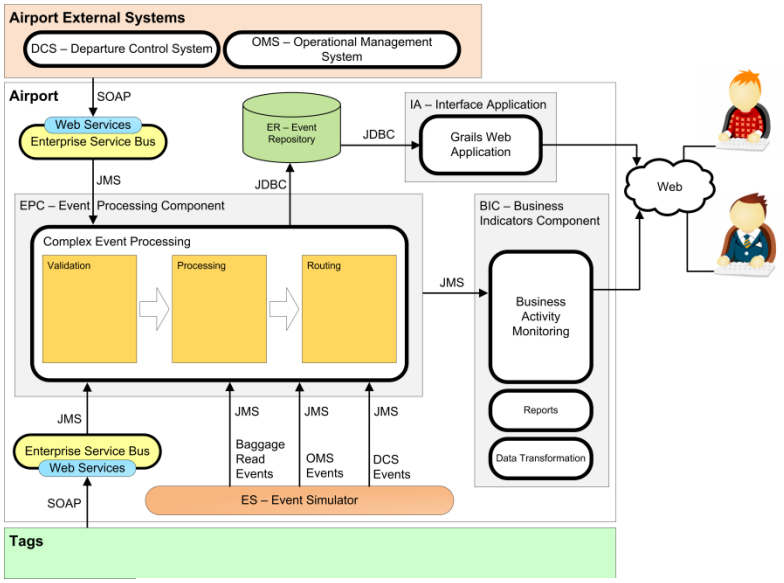

Fig. 2.3. System's overall architecture.

The IA is a Web application that users can use to access information about the baggage handling process (baggage location, flight state, triggered alerts, etc.) that has been stored in the Event Repository by the EPC. As the events processed by the EPC are stored in the ER while they are flowing in the EPC's EPN, the information shown by the IA doesn't have a significant delay comparing to the moment when the events really occurred. Therefore it can be considered, without any prejudice for the user, that the available information presented by the IA is the most recent.

The ESB is responsible for offering a standard interface that allows other components to send baggage read events to the EPC.

The ESB after receiving a baggage read event routes the event to the expected EPC queue.

To realize this, the ESB publishes a Web Service that will be invoked to send read events to the EPC, triggering the delivery of the corresponding XML message to the EPC. The events coming from the airport external systems (DCS and OMS) will also be intermediated by the ESB.

The Event Repository enables the persistent storage of events for accounting and future access. The event persistency is done in the EPC by a single independent thread to avoid a negative impact in the solution performance, since a database access implies a significant overhead.

The Event Simulator sends events directly to the EPC queues, enabling the testing of the system in situations that otherwise couldn't be recreated. The delay between events sent by the ER can be independently defined, allowing more flexible simulations.

3.1 Event Processing Component

Events are processed while they flow through the EPN using different event sending approaches between EPN elements, different ways of processing the events and different ways to communicate with the external entities when events enter the EPC from the Tags layer or when events go out of the EPC towards the Business Indicators Component. Processed events are enriched on the EPN beans that are responsible for correlating various heterogeneous events that flow through the EPN and consequently creating new complex events.

It could be said that every message sent from the solution external systems (messages coming from the OMS, DCS or from the Tags layer) is processed in the following way: there's an adapter that converts the received messages in the respective queues into object events and inserts them in the EPN; then, events are routed to the expected beans to be processed; after that, the bean that received the events delegates the procedure execution to its respective service bean, that encapsulates the business logic; in the end and as the result of procedure execution, the previous bean could send one or more events that will be processed by the next beans in the EPN that will receive the sent events. In the case that one event is sent to a bean responsible for the communication with the BIC, that event will be sent to the Business Indicators Component, refreshing the information reports presented by the CIN.

The EPN elements that take part on the baggage read event processing and the consequent segment time monitoring of each baggage are shown in Fig. 3.2. The `baggageQueueAdapter` converts the baggage reads XML messages on the `BaggageReadQueue` into `ReadEvent` events. Those events are then sent to the `baggageProcessingBean`, passing through the `baggageInStream` and `baggageMonitoring Processor` (that realizes the event filtering, allowing only read events to pass). To process the events, the `baggage ProcessingBean` delegates the needed procedure execution to the `baggageSupportService` service bean. The service bean creates the `Baggage` domain object if the baggage doesn't exist in cache (`baggageCache`). If this was the first read of the baggage, a `FirstReadEvent` will be created to be subsequently sent.

If the referred baggage was associated with a specific flight (because it was previously received by the EPC an event from the DCS associating the bag with an existent flight), an `AssociatedReadEvent` event it's created and the bag state is changed to "under monitoring" (`MONITORING`). Assuming that the baggage is associated, it's possible to know the path that the baggage should travel and therefore the monitoring point referred in the baggage read event it's verified to check if the baggage is in the right path. If the monitoring point doesn't belong to the expected path that will lead the bag to its correct flight, the baggage state is changed to `WRONG_PATH` and a `BaggageWrongPathEvent` alert event is created to be sent afterwards.

If the bag isn't already associated with some existent flight, the baggage state is changed to "not associated but under monitoring" (`NOTASSOC_MONITORING`) and it's created a `NotAssociatedReadEvent` read event. If the monitoring point where the bag was read is considered as an end point (entrance of a plain), it's created an alert event that represents the occurrence of a bag entering a plane without being associated with any flight (`BaggageNotAssociated Event`).

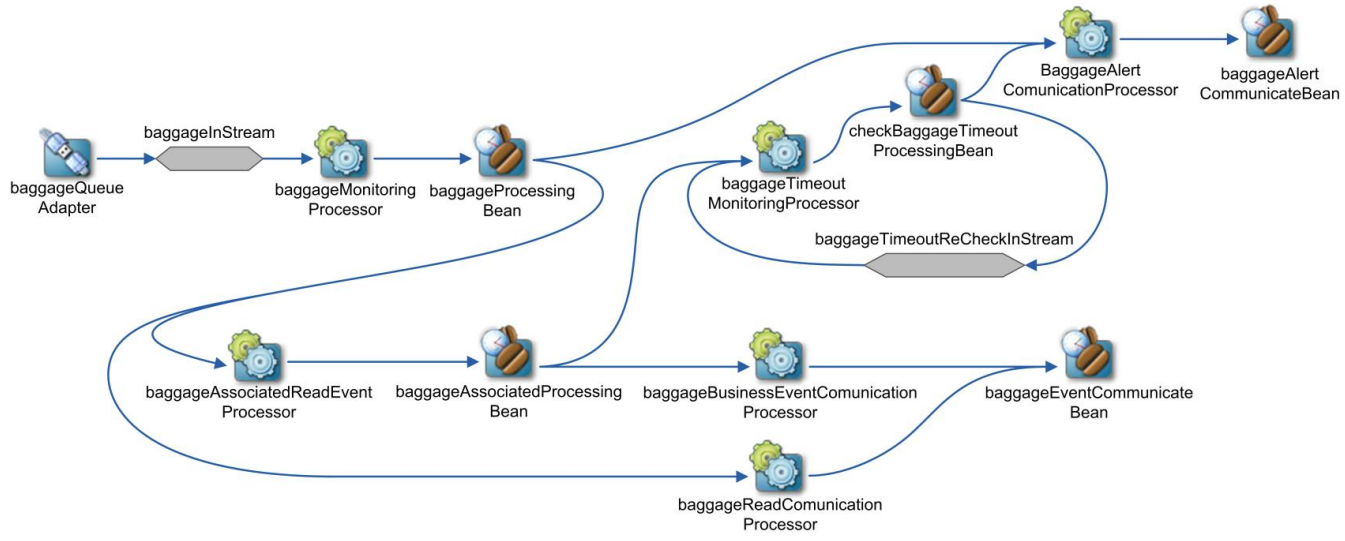

Fig. 3.1. View of the event processing network responsible for monitoring the segment timeouts for each baggage.

After the business logic execution by the baggageSupportService, the events created before are sent by the baggageProcessingBean to the following beans in the EPN: to the baggageAlertCommunicate Bean, in case that alert events need to be sent (BaggageWrongPathEvent or BaggageNotAssociated Event) that should be sent to the BIC, using the BaggageAlertCommunicateBean; to the baggageEvent CommunicateBean, in case that baggage reads need to be sent to the BIC; and finally, to the baggageAssociated ProcessingBean, if the baggage referred in the read events is associated with a flight.

When an associated baggage read event (Associated ReadEvent) arrives at the baggageAssociated ProcessingBean, this bean delegates the event processing to the baggageAssociatedService service bean. The time that each baggage takes on each segment is also monitored on this bean, since any read event here received is related to a baggage associated and therefore it's possible to know in this bean which path should the baggage travel and what are the maximum times for each path's segment.

To monitor the times that each bag takes in the different path's segments, it was used a cyclic event emission mechanism in the EPN that can be seen in Fig 3.2, using two EPN elements: the baggageTimeoutMonitoringProcessor processor and the checkBaggageTimeoutProcessing Bean bean. When an AssociatedReadEvent arrives at the BaggageAssociatedProcessingBean a time check event (TimeoutCheckEvent) is created containing the actual location of the baggage referred on the received event and the maximum time associated to the segment where the bag is. The created time check event will be sent to the checkBaggage TimeoutProcessingBean, passing through the baggage TimeoutMonitoringProcessor. The query that defines this processor is: SELECT * FROM TimeoutCheckEvent retain 10 seconds. This means that every time check events sent to the time monitoring bean will be delivered to it after being retained 10 seconds in the referred processor.

When the event arrives to the checkBaggageTimeout ProcessingBean, this bean delegates to the timeouts Service the time check procedure (if the maximum time was already reached). If the maximum time wasn't reached yet and the baggage is still in the same segment, than the TimeoutCheckEvent received is sent again to the baggageTimeoutMonitoringProcessor. If the baggage location didn't match the monitoring point referred in the time check event when the maximum time is verified, the time check event is ignored, since the baggage was already read in other monitoring point that belongs to other path's segment.

If the maximum time was reached and the baggage was still in the same place referred by the time check event, the system infers that the baggage took more time than the expected to travel the actual segment and therefore should be sent to the baggageAlertCommunicateBean an alert event representing the excessive delay of the baggage on the actual segment (BaggageTimeoutEvent).

Executing the baggage time monitoring as described above the efficiency and precision are increased since the need for a thread in infinite cycle checking the times of every baggage is eliminated and the delay between each time check of each baggage is defined independently for each baggage.

3.2 Other Components

The Business Indicators Component, the Interface Application and the Enterprise Service Bus that compose the solution are the remaining components of the solution architecture that will be here described.

3.2.1 Business Indicators Component

The Business Indicators Component (BIC) was developed using Oracle Business Activity Monitoring (BAM) tool and has the main objective of showing to the baggage handling personnel real-time reports with information about distinct aspects of the handling process: level of accomplishment of Service-Level Agreements (SLAs), lists of event occurrences that might happen during the process execution (baggage delays in some segments of the process, luggage detected in wrong places or in wrong paths, etc.). To accomplish that, the messages sent by the EPC representing all sorts of occurrences in the baggage handling process (baggage reads, baggage located in the wrong path, baggage unloaded, etc.) are received by the BIC through the JMS topics alertTopic, flightTopic e baggageTopic. These topics contain, respectively, alert messages (e.g. a not associated baggage entered a specific plane), messages concerning flights (e.g. a new flight has been registered in the system) or baggage related messages (e.g. bag reads, association between bags and flights, etc.).

All the accesses to data in Oracle BAM are done using a Data Object. Data Objects are responsible for putting all accessed data in memory (on the Oracle BAM's Active Data Cache) before it can be accessed, reducing object access latency. As the JMS topics are a source of data, therefore each one have to be accessed through the respective Data Object. The following Data Objects hide the details of accessing the topics above described: alertsMessages, flight Messages e baggageMessages. Each Data Object can be seen as a table, with a set of fields that contains several rows.

After receiving the messages from the EPC, the corresponding Data Object rows are processed by a set of chained procedures that manipulate the data encapsulated in the messages until it can finally be used in the reports about the baggage handling process. Each set of chained procedures is called a *plan*.

The dashboards presented to the user that compose the BIC are called *reports* in the Oracle BAM terminology. Each element presented in the reports (charts, lists, gauges, etc.) uses the information contained on the Data Objects to show the current state of the handling process. During the reports elaboration using the Active Studio bundled in the Oracle BAM tool, each report element is configured by defining what kind of information will be presented to the user: the actual measure of a specific SLA, the average number of delayed bags by flight or the number of baggages not associated and incorrectly loaded on a specific day, etc.

3.2.2 Interface Application

The main objective of the Interface Application (IA) is to allow passengers and handling process personnel to easily access information about the current baggage state (like the location of a baggage, the monitoring points where the bag was already read, if the bags are already on plane, etc.). By allowing passengers to

immediately access information about their luggage state, passengers' unconfidence and disbelief towards the baggage handling process can be reduced. Constant and active monitoring all the baggages that passes through the baggage handling process allows early detection of process errors before those errors can have more severe consequences to passengers. Also, as the passenger can always have information about its luggage, the disbelief about the process can be mitigated.

The IA was developed using Grails [10], an *open source* development Framework for Web applications supported by Groovy, an OO dynamic language Java based that when compiled results in native Java VM *bytecode*. This Framework was chosen based in the fact that the IA wasn't the primary objective of this work and therefore there was the need of tools that allowed fast development of a Web application without the hassle of configuring an application server, deploying a database, etc.

The Interface Application allows also baggage handling personnel to access the list of triggered alerts grouped by a pre-defined criteria (by flight, time slice, etc.) or instead it allows to directly access the information stored in the Event Repository (used to create the information consulted by passengers). It's also woth noting that the IA doesn't work in real-time, i.e., it works only as an application that allows the access to historical data contained in the Event Repository, even if the baggage reads are shown by the IA in the moment they're stored in the ER.

It's possible to define access permissions to different user roles on the IA using the implemented user control access mechanism, allowing the restrain of information access to users that do not have the needed credentials (passengers shouldn't know the existence of any triggered events, as an example). The user access control was implemented using the Spring Security Plug-in available to Grails.

3.2.3 Enterprise Service Bus

The Enterprise Service Bus enables the definition of a set of standard interfaces to each heterogeneous solution component, linking the created interfaces with the original interfaces of each component. In addition of being the intermediary between different solution components, it can also convert the messages sent between each component, so that the receiver can correctly understand the message.

In the solution diagram (cf. Fig. 3.1) the Tags layer communicates with the EPC by means of a Web Service invocation. This layer is responsible for managing and capturing all the baggage read events. The RFID baggage reads, already filtered and normalized by the Tags layer are then sent to the EPC. The Web Service created in the ESB is shown to the Tags layer as the interface that will take the baggage reads to the EPC and therefore, it's Web Service invocation will represent the read a specific baggage.

The information contained in each Web Service invocation is composed by the baggage's LPN, the read timestamp, the code that represents the location where the tag was read and finally by the expected time that the baggage flight will take off. This information is stored on the baggage tag when the tag is emitted and attached to the baggage, namely when the baggage check-in is done (the time flight can only be known during check-in). The flight time on the tag is needed because there's the chance that the same baggage could have two RFID tags (one of them from a previous flight) and therefore the most time efficient way of distinguish the correct tag is to compare the flight time of each tag and to ignore the tag that has the older flight time.

After the WS invocation, the `BaggageReadRoutingService` (one element that belongs to the ESB) routes the information sent by the invocation to the `BaggageReadJMSAdapter` (the other ESB element) that creates the XML baggage read corresponding message and consequently inserts the message in the respective JMS queue (`BaggageReadQueue`). The EPC will receive the baggage reads from the referred JMS queue.

4. SYSTEM VALIDATION

In this chapter it is described the validation approach for evaluating the proposed system.

In order to validate the designed application during its development, it was necessary to execute unit and integration tests created to guarantee the correction of the developed classes that encapsulate the system's business logic and the correct interaction between the various system components. The system was also simulated using an event simulator developed in the scope of this work.

4.1 Development Tests

The developed prototype was continually tested using unit tests to guarantee the correctness of each developed unit of code. These tests verified the expected result of each unit of code and were created before the actual development of each unit, using a *test-driven development* approach. The creation of the unit tests responsible to assert the correct behavior of the service beans was done with special care since those beans encapsulate the all the Event Processing Component business logic.

After the successful execution of the unit tests, several integration tests were created to assert the correct behavior of all the solution components as a whole, including the simulation of different departure and arrival scenarios, using messages from every possible source (Tags layer, DCS and OMS).

4.1.1 Unit Tests

The main objective of the created unit tests was to assert the correction of each EPC's unit of code. These tests were made using the unit tests Framework JUnit [11] that enables the easier creation and management of unit tests for Java based applications.

Every service bean has its corresponding test class, in which the business logic correctness of each method is tested. As an example we have the service bean `TimeoutsService` that have the class `TimeoutsServiceTest` as its test class. This test class is composed by unit tests that assert the correct processing of the `TimeoutCheckEvents` that flow through the EPN.

The dependency between many service beans (e.g. the `TimeoutsService` bean needs the `BaggageService` bean to obtain the bags referrend in the time monitoring events) implies the chance of incorrectly detecting an error in one service bean. This can happen because when the tests are being executed, the execution of the service bean on which the tested bean depends concluded with errors, resulting in an error detected on the tested bean, even if there was no error on the tested bean code. To avoid this kind of situations, *mock objects* were used to simulate the expected behavior of the beans on which the tested bean depends. Mock objects reproduce the expected behavior of the real service bean. They are created in the referred test classes for each service bean needed by the tested bean, respecting the interface of each service bean needed. With this kind of approach, it's possible to

test each unit of code isolately without requiring the correct behavior of each existent dependency.

4.1.2 Integration Tests

The integration tests pretend to verify that all architecture components interact correctly with one another to deliver the expected behavior of the monitoring system. With that in mind, it different tests were created that demand the simultaneous participation of different solution components to assert the expected integration between architecture components.

Before the execution of tests that demanded the simultaneous participation of every component, some tests that only needed a subset of the existent components were executed, to detect possible errors with an increased granularity. To test the correct behavior of the Web Service delivered by the ESB (namely, the placement of a XML message in the correct JMS queue with the expected information introduced using the previous Web Service invocation) an *open source* tool (soapUI) developed to test Web Services was used. After the invocation of the Web Service with the referred tool, it was verified that the corresponding XML message arrived at the EPC adapter responsible for converting baggage read messages contained in the `BaggageReadQueue` into `ReadEvents` that can be injected into the EPN. Therefore, the correctness of the ESB behavior, the link between ESB and the EPC and the conversion from XML messages to read events by the EPC adapter was verified.

Tests were also created to guarantee the correct interaction between the EPC and the Event Repository. The database access details were hidden by each Data Access Object (DAO) created in the EPC. These DAOs (e.g. `FlightDao`, `BaggageDao`, etc.) are beans that encapsulate the needed logic to access the data objects stored in the database. The tests that assert the correct integration between the EPC and the Event Repository are the unit tests developed to assert the correction of each DAO. This tests not only verify the connection to the database but also verifies the correct correspondence between objects and their respective attributes in the EPC's domain model and the corresponding database tables and columns.

4.2 System Simulation

To enable the simulation of the developed prototype in the most realistic possible scenarios, it was created an Event Simulator (ES) that interacts with the EPC using JMS (cf. Fig. 3.1). As the implemented solution only receives events via JMS queues (independently of the messages source, namely the Tags layer, the DCS or the OMS), the Event Simulator only have to place each simulated message in its corresponding JMS queue using the message source as routing criteria. From the EPC point of view, it's impossible to distinguish between events coming from the simulator or events coming from the real external systems or from the Tags layer, since the syntax used in the messages sent by the ES respect exactly the same syntax used in the real messages.

To assert the expected system behavior in real situations, various departure and arrival scenarios were created. These scenarios were created with the objective of executing the simulation of the system in similar situations to the ones met in a production environment. The ES was responsible for parsing each scenario (described using a XML file) and to send the corresponding events to the EPC. As a result, it was expected that the correctness of the following items was confirmed: the EPC behavior; the events persistency in the ER; the presentation of the monitoring history by the IA; and finally, the presentation of the business indicators reports that contained information about the SLAs measurements, detailed information about the detected exceptions on the baggage handling process and also information about the executed flight operations grouped by each flight.

In the arrival scenarios, the tests were started by only simulating a single flight arrival with all baggages being correctly delivered. Then, it was associated more than one valid path to the used flight to exercize the situation when some bags are delivered to the baggage carousel using different paths (e.g. over-sized baggage). After this test, the read events sent by the ES were changed to trigger various timeout alerts because some bags took excessive time in different segments. Finally, read events in monitoring points that didn't belong to the correct path were created with the objective of triggering wrong path alerts because the bags were detected in incorrect paths. It was also verified during all simulations that the events representing a bag being unloaded from a plane, a baggage delivered to the baggage carousel and a baggage detected at the airport exit were correctly sent by the EPC to the Business Indicators Component.

It's worth noting that the prototype here presented consists in a demonstration that will be delivered to the client of the project where this work is inserted as a proof that it's possible to monitor in real-time the baggage handling process, obtaining detailed information about the process and about each processed baggage. The prototype will also be tested in its production environment (implying the installation of each component in production servers and doing the detection of each baggage on the various monitoring points with real RFID readers connected to the Tags layer) in the last days of October, 2009 and therefore it won't be possible to include the results obtained in this work.

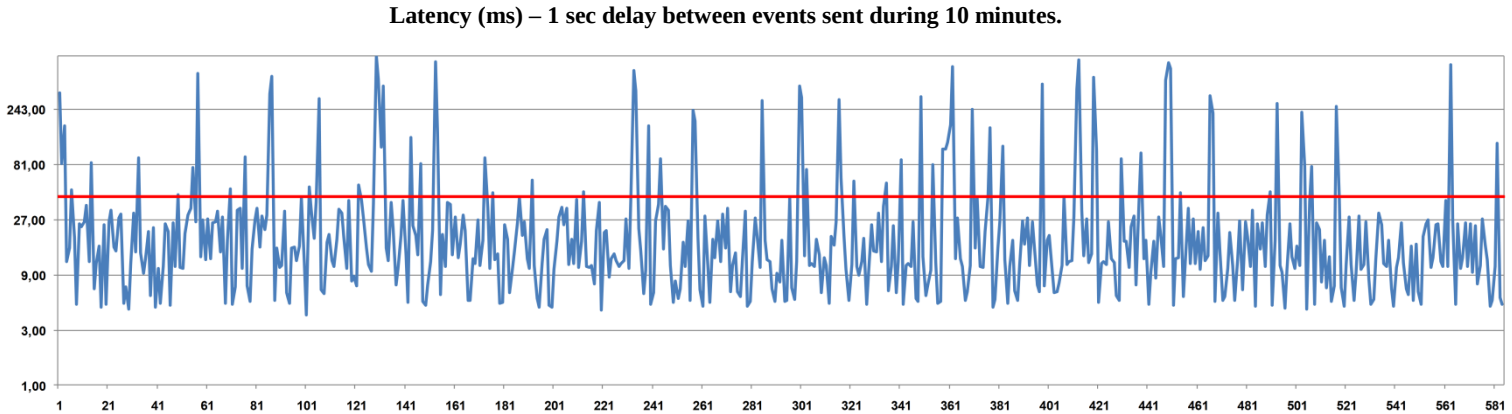

Fig. 4.1. Observed latency while the benchmark events are sent. The average latency is in red.

Despite the lack of real testing results, the prototype was developed and tested in a virtual machine (VM) based on Windows 2003 hosted on a machine powered by a Core 2 Duo T7300 2Ghz processor with 4GB RAM also installed with Windows Server 2003, that doesn't have the computation power of a server but can ultimately illustrate the implemented functionalities. Therefore, the prototype installed in the referred VM was used to execute all the described tests and consequently the expected results were obtained without being hassled by the VM low performance.

4.2.1 Event Processing Component Benchmark

As said before, it wasn't possible in a timely manner to test the prototype in its production environment. Because of this, a benchmark executed by Oracle [12] was studied to verify the event processing efficiency of the Oracle Complex Event Processing.

In the referred benchmark, the latency of the events processed by CEP is measured in the context of an application that works with stock markets with one event processor executing two different querys on the received stock ticks. With an event injection frequency of a million events per second, the application used in the benchmark shows an average latency of 67.3 microseconds and a maximum latency lower than 25 miliseconds. In this benchmark, the Oracle CEP was installed in a Intel Xeon server with 4 Intel X7350 Quad-Core 2.93GHz processors (each with 4 cores), 32 GB RAM and running a Red Hat Enterprise Linux 5.0 32 bits version.

In the Event Processing Component of the solution, an event passes through 9 EPN elements at max (assuming that the time that each baggage takes to travel one segment is only monitored once). Those 9 EPN elements are 4 event processors, one adapter and 4 beans. Knowing that the event processors used in the solution are only responsible for letting pass the expected events, the latency that each processor implies is negligible when compared to the time that each event take in the various EPN beans, since those beans include all the business logic.

To measure the event processing latency of the prototype installed on the virtual machine used in the development of the solution, it was created a synthetic benchmark that involves the emission of a certain number of read events that will result in business events (that have to be consequently sent to the BIC). The time when the read event was injected in the EPN and the time when the corresponding business event was sent to the BIC is registered to obtain the latency for each emitted event.

The test consisted in the generation of read events using the Event Simulator during 10 minutes with a 1 second delay between each event, resulting in the emission of nearly 600 events. The measured average and maximum latency can be seen in Table 4.1. The time was measured between the `ReadEvent` creation in the `baggageQueueAdapter` after obtaining the respective XML message from the queue and the moment when the processed read event was sent to the BIC in the `baggageEvent CommunicateBean`. In Fig. 4.1 it's possible to observe the measured latency as the events are received in the EPC and consequently are sent to the BIC with the average latency represented in red.

Average latency	42.54 miliseconds
Maximum latency	696.71 miliseconds

Table 4.1. Average and maximum latency observed in the executed benchmark.

4.3 Non-Functional Requirements Validation

In the following table the non-functional requirements defined for the developed system are presented along with the way they were accomplished in the proposed architecture.

Non-Functional Requirement	How it was accomplished
Security	User access control mechanism used on the Interface Application.
Modifiability	Development ruled by design patterns and the developed source code is well documented.
Testability	Development of an event simulator and the use of a testing framework to enhance the testing procedure.
Modularity	Use of standard mechanisms of communication between all architecture components (Web Services, JMS). The syntax of the messages sent between components is also well defined.
Latency	On the executed benchmark the maximum latency observed was 696.71 ms (0.7 sec).
Adaptability	It's possible to adapt the system to any airport by configuring the airport paths without changing any functionality of the system.

5. SUMMARY AND CONCLUSION

In the scope of this work, it was specified and consequently developed a monitoring system applied to the baggage handling process that exists in any airport.

It was needed that the proposed system didn't impose the change and/or the adoption of any specific baggage handling system, enabling the modular installation of the solution in any baggage handling process. The way how the solution interacts with the airport systems and the functionalities implemented respect all resolutions required by IATA applicable to the solution context. Therefore, the seamless adaptation to any airport of the proposed system is guaranteed.

The implemented prototype, even if it was only a demonstration of the technology used, should support from its first stable version the expected number of monitoring points and the expected event flow that will exist on the airports where the prototype will be initially tested (Ponta Delgada, Horta, Santa Maria and Flores airports). The tests done to the delivered prototype, even if they weren't executed in machines equivalent to the ones that will be used in production, guarantee in part that the developed prototype will support the event load expected in the production environment.

This system increases the monitoring detail when compared to current monitoring systems because it allows the individual tracking of each processed bag, since the information used is obtained by reading the tag placed in each bag allowing the detection of exceptional situations happening only to a single bag. This reduces the reaction latency on this kind of situations, allowing smaller correction times.

This work was focused mainly on the event processing theme and therefore the tool used to develop the Event Processing Component was thoroughly studied. The Business Indicators

Component is present in the delivered prototype primarily as a proof-of-concept that it's possible to measure various business performance indicators using the information obtained by the Event Processing Component. Having said that, it's also important to note that the information presented by the Business Indicators Component consists integrally in the performance indicators required by the client of the project in which this work belongs to.

The tests that will be executed to the delivered system in the production environment weren't executed in a timely manner for the inclusion of the obtained results in this work. Despite the lack of practical results, the system was simulated in common departure and arrival scenarios and some load tests were also executed to enable the verification of the correctness and the processing efficiency shown by the delivered system.

Observing the objectives proposed for this work, it can be said that they were accomplished since the context in which the baggage monitoring problem is inserted and the various works previously realized in different areas that relates somehow with this work's theme were comprehensively studied; it was proposed a solution that in theory, presented an architecture that accomplished all the pre-defined requirements; it was developed a prototype that realizes the different components introduced in the architecture definition; measures were taken to ensure the correctness of the created system; and finally, all the developed work was written in a structured and methodic way enabling readers to comprehend the full scope of this work.

5.1 Future Work

Despite the accomplishment of the objectives proposed to the developed work, various questions were noticed during the development of this work that could be, if more time was available or if the work's scope was changed, improved and/or developed in a more detailed way.

The developed system doesn't have yet fault tolerance mechanisms, even if the elements needed to implement those type of mechanisms are already present in the solution architecture (e.g. it's possible to add another Event Processing Component acting as a redundant component or an active parallel component and it's also possible to use the persistency mechanisms included in the JMS queues used in the system). If the referred measures were taken, the availability of the system could be increased, reducing the time lapses in which the baggage handling system is not monitored. The use of a parallel EPC is supported by the possibility of concurrent accesses to a JMS queue. In this case, both Event Processing Components would share the JMS queues and the information (about bags, flights, etc.) stored in the Oracle CEP caches. To accomplish the cache sharing between EPCs, it can be used a distributed cache solution from Oracle [13] (Oracle Coherence) that allows the transparent management, synchronization and sharing of caches. Using a parallel Event Processing Component the availability and the total processing power of the solution would be increased.

Using the distributed cache mechanism introduced before will also be possible to remove the Web Service invocation as the method used to send read events from the Tags layer to the EPC, since the Oracle Coherence caches have simultaneously Java and .NET APIs that will enable the Tags layer (that uses .NET) to directly place the read events in the same cache that the EPC (Java based) will read to obtain the events that will be processed.

Other situation that can be improved is the presentation of the traveled path by each baggage when this information is accessed using the Interface Application. The monitoring points list shown to the user, despite of the description in each point, isn't completely explicit to the common passenger. Therefore, it could be used a Geographic Information System to map the monitoring points of the presented path in a diagram that represents the physical displacement of the airport. Using this approach, the passenger will have visual references that could aid the comprehension of the path traveled by its bag.

It could also be created in the future a new component in the solution architecture responsible for notifying the passenger about relevant events that have been detected with its luggage. The passenger should be able to subscribe to each available event type and the notifications would be delivered through different channels: e-mail, SMS, twitter, etc. The created component could use the EPC asynchronous event delivery, using a new communication bean on the EPN equivalent to the already existent communication beans that checked for each received event, if the event should be communicated to the passenger. As an alternative to the above method, it could also be used the Message Center included in the Oracle BAM architecture that allows the notification of different receivers using pre-defined criteria to decide if should send or not each notification.

Actually, the exceptional situations are detected using correlation of various types of events in the beans integrated in the EPN of the EPC and consequently the baggage handling personnel is alerted to the detected situation via reports presented by the Business Indicators Component. Only after being warned about the current situation the baggage handling team reacts accordingly, trying to correct the situation in the smallest possible time. In the future, the correlation of the various alerts sent to the BIC could be made to enable the automatic trigger of correcting actions, without any human intervention.

6. REFERENCES

[1] I. A. T. Association, "Baggage improvement programme, simplifying the business," International Air Transport Association, Tech. Rep., 2008.

[2] T. Vogel, "Technical system specification to rfid-enable the baggage sorting system at zurich airport," Master's thesis, Swiss Federal Institute of Technology Zurich, 2007.

[3] I. A. T. Association, *Passenger Services Conference Resolutions Manual*. International Air Transport Association, 2008.

[4] O. Etzion and P. Niblett, *Event Processing in Action*. Manning, February 2010.

[5] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom, "Stream: The stanford data stream management system," Stanford InfoLab, Technical Report 2004-20, 2004. [Online]. Available: http://ilpubs.stanford.edu:8090/641/

[6] P. Purich, *Oracle CEP Getting Started*. Oracle, May 2009.

[7] Z. Ting, O. Yuanxin, L. Chao, and X. Zhang, "A scalable rfid-based system for location-aware services," *IEEE*, 2007.

[8] Z. Ting, O. Yuanxin, and Y. He, "Traceable air baggage handling system based on rfid tags in the airport," *Journal of Theoretical and Applied Electronic Commerce Research*, vol. 3, 2008.

[9] Z. Ting, X. Zhang, and O. Yuanxin, "A framework of networked rfid system supporting location tracking," *IEEE*, 2006.

[10] Grails. (2009, October) [Online]. Available: http://grails.org/

[11] V. Massol and T. Husted, *JUnit in Action*. Greenwich, CT, USA: Manning Publications Co., 2003.

[12] Oracle. (2008, September) Oracle complex event processing performance. [Online]. Available: http://www.oracle.com/technology/products/event-driven-architecture/collateral/cepperformancewhitepaper.pdf

[13] Oracle. (2009) Oracle coherence data sheet. Fornecido pela Oracle. Consultado em Outubro de 2009. [Online]. Available: http://www.oracle.com/products/middleware/coherence/docs/oracle-coherence-data-grid-datasheet.pdf