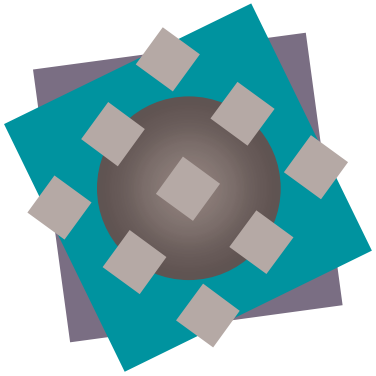


WareLite Real Time Process Design & Deployment Methodology: a Delivery Tracking scenario

Index

1 PREFACE

Traditionally, the deployment of Software supporting event-driven business processes has been a multi-step approach, where each step away from the initial definition of a process represented a compromise between the original business idea and its actual implementation.

With WL BOSS, WareLite closes the gap between the definition of a new business strategy and its deployment.

WareLite BOSS provides a solution design & run-time environment where Business Logic and System Infrastructure are entirely separated, thus giving Enterprises the opportunity to focus on business requirements, rather than on systems' constraints.

As all non-business related process management issues – data persistence management, determinism, transaction management and scalability – are dealt with by WL BOSS infrastructure, business processes can be simply designed and immediately implemented using WareLite **Real Time Process Design & Deployment** methodology (**RTPD^2**).

The first step of WareLite **RTPD^2** methodology is **Scenario Description & Analysis**. Once a desired (TO BE) Business Scenario is identified by the business owners, business consultants will describe it in terms of **Stakeholders**, their **Interactions** with the enterprise and the Business Processes – or **Services** – that support such Interactions. The required Quality of Service (**QoS**), like availability (e.g. 24/7) and throughput (e.g. number of transactions per second), will also be defined.

As the first Scenario Analysis iteration is completed, the **Process Design & Deployment** phase of WL RTPD^2 begins. During this phase, technical consultants will translate the business definitions of Stakeholders, Interactions, Services and Quality of Services into **Roles (XML), Objects Classes (XML), Business Rules (C++)** and Grid **Domains**, implementing a fully working Business Scenario prototype.

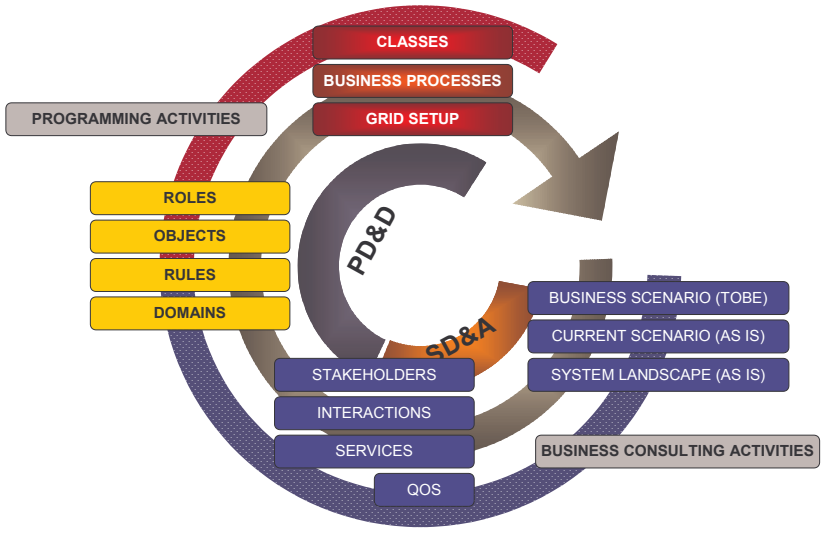
By observing the working prototype, business owners and business consultants will be able to refine the TO BE scenario, thus starting a new design & deployment cycle that will improve the software prototype.

Once the Business Scenario prototype is completed, it is ready to enter the production phase via simple system configuration – as WL BOSS takes care of transaction management, determinism, persistence and scalability, there is no need for further development activities.

The running business processes can at any time be easily modified and enriched by adding or changing roles, classes of objects, business rules.

All roles, classes of objects and business rules thus developed can be reused in any other business process running on WL BOSS. All data objects belonging to the classes thus developed can be freely accessed by all the business rules running on WL BOSS.

Thus, teams working separately on the development of different Business Scenarios will create business processes that will natively be 'integrated', being made of reusable components that can natively access and modify all the enterprise data.



WareLite Real Time Process Design & Deployment methodology

This document provides an introduction to WareLite Real Time Process Design and Deployment methodology through an application example.

The application example described in the following pages is Real Time Delivery Tracking.

2 SCENARIOS

2.1 Company Targets: TO BE Scenarios

2.1.1 PHASE 1 TO BE Scenario– Customer Services Provision

A carrier company keeps track of a moved 'package' en route to its final delivery destination. The information about package's position must be available to the company's customers.

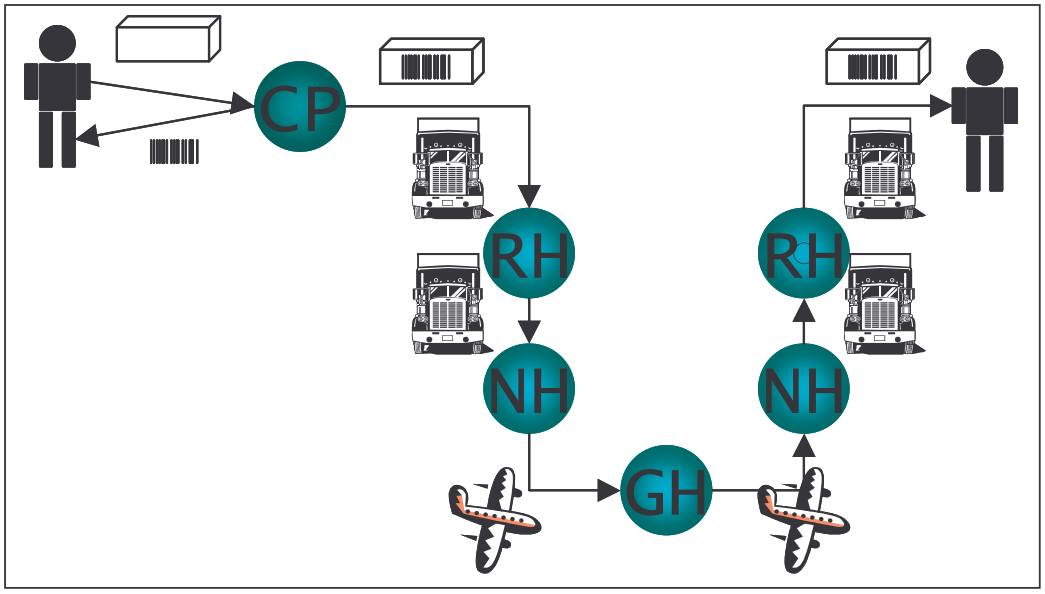
2.1.2 PHASE 2 TO BE Scenario – Enterprise Resources Management

The same information (package's position) is used to monitor the volume of goods transiting in all the company's hubs. With this extension the carrier aims at improving capacity planning. The same information is also used to calculate the average cost of delivering a package (the cost is initially expressed in number of hubs touched by a package).

In the following example, Phase 2 is an extension of the business processes defined for Phase 1.

3 'TO BE' SCENARIO DESCRIPTION

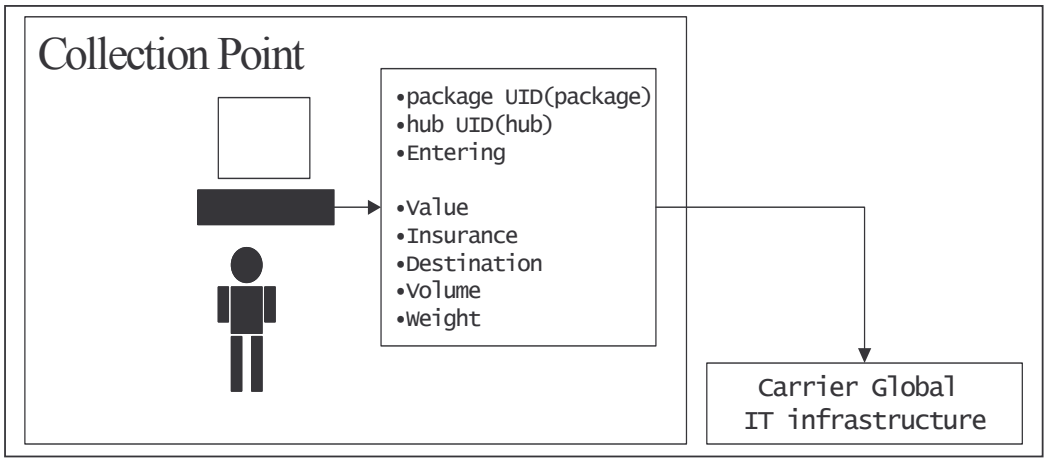
Every package is moved through a set of hubs owned by the carrier. For instance, the customer takes a package to a carrier's collection point (**CP**). From there the package is moved to the regional hub (**RH**). From the regional hub the package is moved to the national hub (**NH**), to a global hub (**GH**), to a national hub, to a regional hub and eventually it is delivered. The final route (expressed in hubs) of a package depends on the specific logistic adopted by the carrier.
The figure above shows an example of a route followed by a package (or item). In summary, the customer takes the package to the Collection Point. The Collection Point gives the customer the package code and 'wraps' the package.



Wrapping is the process with which the collection point weights the package, selects a carrier standard 'envelop', so that the volume is known, and associates the package with the code that has been given to the customer. For instance, the Collection Point attaches a sticker (bar code or RFID tag) to the wrapped package. The Collection Point, of course, collects other information such as the address of the recipient (destination), the package value and its insurance value. Also, the Collection Point selects a tariff (e.g. based on the desired quality of service) and typically presents an invoice to the customer.

3.1 Collection Points

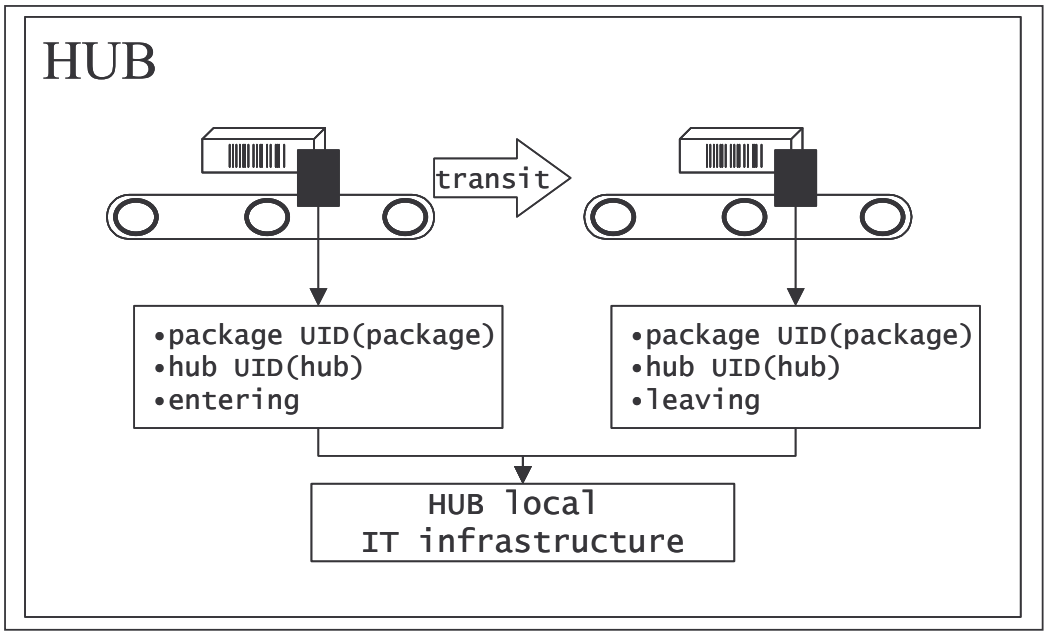
The Collection Point is thus a special hub that initiates the service of the carrier. Also the Collection Point is particularly important because it collects all the main properties (unique id, value, origin, destination, weight, volume, etc.) that identify the package. We assume that all this information is 'transmitted' immediately to the global IT infrastructure of the carrier, as shown in the picture. In case of RFID tags, this information is associated with the Serial Number of the tag.



We also assume that, as in the case of any other hub, the Collection Point hub is capable of notifying the global IT infrastructure as soon as the package leaves (e.g. as soon as the package is collected by the local vector). In case of RFID, this notification can be automated by the presence of antennae on the outbound gate of the facility. From the Collection Point (the first hub) the package is sent to a National Hub and successively to a Global Hub. From there the package, via another national hub and via a regional hub, will eventually arrive to the recipient. The recipient stakeholder can be thought of as the last hub. Please note that several carriers already have a set of custom PDAs so that they can electronically collect the signature of the recipient. Such 'event' is particularly important. By adopting such technology the carrier can more easily solve any disputation case and better monitor the overall performance of any local carrier involved in the delivery chain.
Also, depending on the extra services the carrier might want to provide, the information provided by the collection point can be enriched. For instance, the collection point might provide information about the customer e-mail, so that as soon as the item crosses the boundaries of a hub the customer is automatically notified by the carrier in a push fashion. Such kind of extra services might be free of charge or they might generate extra revenue.

3.2 HUBS

Each HUB is equipped with a set of devices (e.g. barcode scanners or RFID antennae) so that the carrier can retrieve the following information: time of package entering the HUB; time of the package exiting the HUB. Also, any device within the HUB 'contains' the information about its location. In summary, the carrier has in place all the infrastructure needed to know in which HUB the package is at any time: as soon as the package enters a HUB a barcode reader or a RFID antenna scans the package wrap and sends a properly packaged message to the local IT infrastructure; as soon as the package leaves the HUB a barcode reader or a RFID antenna scans the package wrap and sends a properly packaged message to the local IT infrastructure. We assume that the data frame sent by a barcode reader or a RFID antenna already contains the following information: unique identifier of the package, unique identifier of the hub, time and a tag saying whether the package is entering the hub or leaving the hub. The 'latency' a package has within a hub is depicted as 'transit'.



Once the barcode scanners/antennae have delivered the proper information to the local hub IT infrastructure, it is then responsibility of such infrastructure to deliver the same information to the carrier's global IT infrastructure. In this very moment we do not want to make any assumption about the implementation of local IT infrastructure. The only assumption we are making is about the capability to manage the information we have represented and to deliver such information to the global IT infrastructure of the carrier. At the end of this scenario we will show a possible system implementation.

3.3 Customer

The customer interacts with the enterprise in two main ways: he/she takes a package to a collection point and retrieves information about the delivery status (or position) of a package. For the sake of this discussion we assume that any kind of relationship the customer has with a collection point is mediated by the collection point itself: if a customer consigns a package to a collection point it will be responsibility of the collection point to notify the enterprise of such event. Instead, whenever the customer wants to retrieve information about the delivery status of a specific package he/she will directly initiate an interaction with the enterprise. Typically the carrier might expose a Web based interface to provide the customer with such pull-oriented service. Other more advanced services are possible. For instance the customer might have bought an extra service so he/she will be notified (via e-mail, SMS or any other possible mean) whenever a package changes its position (push-oriented service).

3.4 Destination (or Recipient)

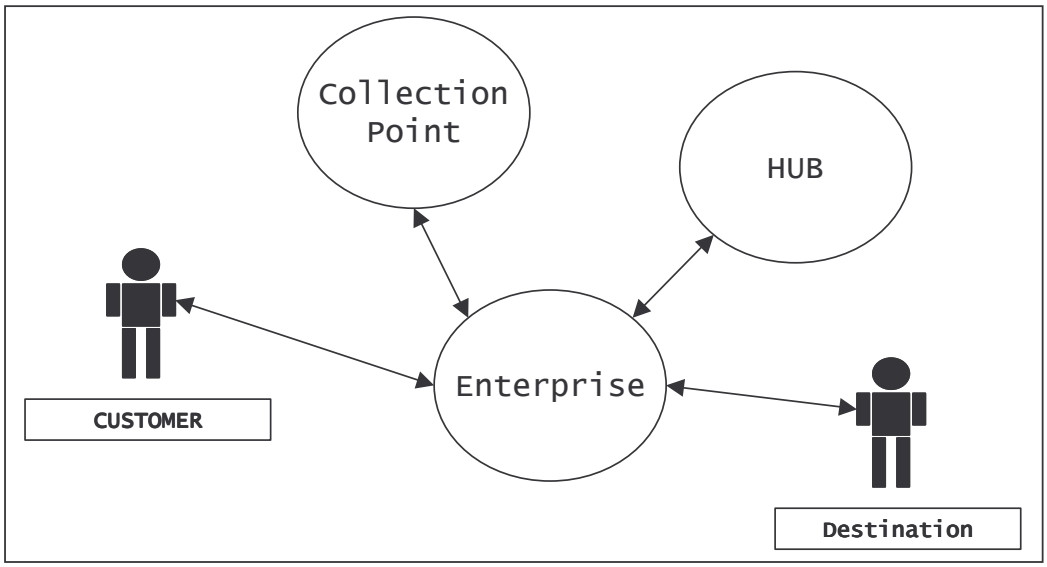
The recipient is the final destination of a package. Very likely, in reality, the recipient will be a person associated with an address. Thus a recipient will interact with the enterprise by accepting or rejecting a package. How is this kind of process usually implemented? Very often (if not always) it is responsibility of the local carrier (or last mile carrier) to collect a signature or – somehow - to notify that the package has been delivered to its final destination. For instance the carrier might collect a signature using a custom PDA type device or it might have other means of providing the information required to fully qualify a final delivery event. In any case – technically - the local carrier will have to impersonate the recipient or it will have to provide the recipient with the proper interface to notify the enterprise of the event ‘final destination’. In our scenario, the last mile carrier will impersonate the recipient.

4 ANALYSIS & DESIGN

4.1 Analysis: Stakeholders -> Roles

Using WL Real Time Process Design & Deployment methodology let's first of all identify the actors of this scenario. We need to answer the question: Who are the stakeholders that will interact with/within the enterprise?

The stakeholders we have identified so far are represented in the following picture.



Based on such representation we can immediately define such stakeholders by creating the proper set of WL BOSS Roles:

- CUSTOMER
- COLLECTION POINT
- HUB
- DESTINATION

In our representation there may be other Stakeholders, such as VECTORS that for the sake of this discussion we do not keep in account.

Now that we have created the first set of roles, let's try to formalize how such Roles will interact with the enterprise.

4.2 Analysis: Interactions

We will now describe the interactions that each Stakeholder has with the Enterprise

4.2.1 Stakeholder: Customer

The customer delivers a package to the carrier's Collection Point.
The customer wants to be able to know where the package is en route to the destination. In order to retrieve such information the customer notifies the enterprise with a message containing:

- package unique identifier

Remember that the package's unique identifier is generated during the wrapping process at the collection point and it is given to the customer.

4.2.2 Stakeholder: Collection Point

The collection point notifies the carrier that there is a package ready for collection. Such notification contains information about e.g.

- package unique identifier
- collection point unique identifier
- time
- package volume
- package weight
- package value
- applied tariff
- quality of service
- insurance
- destination (or recipient)

Also, in our scenario, the collection point has to notify the carrier as soon as the package leaves the collection point (e.g. as soon as the local vector collects the package itself). Such notification may contain the following information:

- package unique identifier
- collection point unique identifier
- time
- direction(outbound)

4.2.3 Stakeholder: Hub

The hub interacts with the carrier notifying whenever a package enters the hub and whenever the package leaves the hub. When a package arrives to the hub, the hub must be able to notify the carrier (or enterprise) with the following information:

- package unique identifier
- hub unique identifier
- direction(inbound)
- time

When a package leaves the hub the carrier has to be notified with the following information:

- package unique identifier

- hub unique identifier
- direction(outbound)
- time

4.2.4 Stakeholder: Destination (or Recipient)

The destination interacts with the carrier by notifying that the package has been received and accepted. Depending on the degree of sophistication, the information contained in such notification may vary. For the sake of this discussion we will assume the following information is used to notify the enterprise that the package has been received and accepted:

- package unique identifier
- name of the receiver
- time
- acceptance (has the delivery been accepted?)

Now that we have thought about how the Stakeholders (Roles) interact with the enterprise, we must define the first set of business objects meant to represent such interactions. Once we have defined such business objects we will then think about how such 'notifications' will be processed. In other words, now we will define the layout of the events that will be presented to WL BOSS by all the stakeholders/roles that we have identified. Later on we will think about the business processes that will have to be triggered by such events.

Within WL BOSS an event is a business object and a business object is an instance of a class of objects. A class of objects defines the data layout shared by all the objects belonging to the same class. Let us now define the classes we need in order to represent all the interactions that we have identified so far. Doing this means to extend the class dictionary managed by WL BOSS. Once a class is defined, then all the instances (objects) that belong to that class will be available to all the business processes hosted by WL BOSS.

4.3 Design: Interactions -> Business Objects

4.3.1 Business Object: Customer Interaction

Logic: request to retrieve information about my package (where is it?)
Class Name: package_info
Data Layout:

Property Name	Type	Mandatory	Remark
package_uid	Unique Identifier	YES	The unique identifier has been associated with the package during the wrapping process. The unique identifier is mandatory

Persistence: the objects of such class do not require persistence. If a customer sends a request for information, there is no need to 'store' the request itself into a persistence provider (e.g. DB engine). The request has to be processed immediately and its scope does not need to be wider than the scope of the process applied to it.
Role Association: CUSTOMER (the customer can interact with the enterprise by presenting to WL BOSS an object (event) that belongs to the class package_info).
WL_DDL (Data Definition Language) representation:

```
<WL_DDL>
      <CLASS id="package_info">
            <INTERNAL>
                  <OBJ_ID id="package_uid" creation="mandatory"/>
            </INTERNAL>
            <EXTENDED/>
      </CLASS>
</WL_DDL>
```

This XML document formalizes the declaration and the layout of the class 'package_info'. This XML document (WL_DDL) is the input for the tool wl_ddl. Using the tool wl_ddl it is now possible to extend the data dictionary managed by WL BOSS so that all the business processes will have access to (will be able to process) the instances of the class 'package_info'.

4.3.2 Business Object: Collection Point Interactions

As we saw before, the stakeholder 'collection point' interacts with the carrier in two different ways. As soon as the package is received (and as soon as the wrapping process is completed) the collection point has to notify the carrier with the following information:

- package unique identifier
- collection point unique identifier
- time
- package volume
- package weight
- package value
- applied tariff
- quality of service
- insurance
- destination

As soon as the package is collected by the local vector, the collection point has to notify the carrier with the following information:

- package unique identifier
- collection point unique identifier
- time
- direction(outbound)

Here we have to different types of notifications (i.e. 2 types of interactions) that we will represent by declaring two different classes of objects. The first class will be 'new_package' and the second class will be 'package_position'.

4.3.2.1 Object Class: new_package

Logic: as soon as the package is received, the collection point has to notify the enterprise with the information (properties) that identify and fully describe the package.
Class Name: new_package
Data Layout:

Property Name	Type	Mandatory	Remark
package_uid	Unique Identifier	YES	package unique identifier
hub_uid	Unique Identifier	YES	collection point unique identifier
time	Date/Time	YES	the time of when the package has been 'wrapped' or received.
volume	Double	YES	the package's volume (e.g. in cubic inches)
weight	Double	YES	the package's weight (e.g. in grams)
value	Double	YES	the declared value of the package expressed in the collection point's

			local currency
tariff	Double	YES	the tariff applied, expressed in the local currency
QOS	Integer	YES	e.g. a code for fast delivery or normal delivery etc.
insurance	Double	NO	if used, the insured value expressed in local currency. If not used then the package hasn't been insured.
destination	String	YES	address of the destination

Here, again, the layout shown has been simplified for the sake of our discussion. For instance the destination might be a more complex structure containing the full route that the package has to follow up to the destination (depending on the planning system applied by the carrier). More in general the destination can be broke down in country, ZIP, Telephone Number etc. There might also be the need to formalize other information such as preferential time of delivery (morning/afternoon/evening). We are skipping all these details in the attempt to focus on the overall scenario. To increase sophistication, the wrapping process might also collect the customer's e-mail so that asynchronous notification will be possible (e.g. as soon as the package changes its position the carrier user will automatically send a mail to the customer).

Persistence: we assume that this class of objects will only be used to support the interactions between the Collection Point and the Carrier. This class will thus only be used to encapsulate the information 'sent' by Collection Point to the Carrier. However, we might want to apply a different logic. For instance we might decide to give persistence to the whole new_package class so that any instance of this class presented by the collection points would be automatically stored into the persistence providers managed by the WL BOSS. By doing this, all the information about the sender and the recipient would automatically be stored into an underlying RDBMS, possibly simplifying any further implementation activity.

Role Association: COLLECTION POINT (the collection point can interact with the enterprise presenting to WL BOSS an object (event) that belongs to the class new_package).

WL_DDL representation:

```
<WL_DDL>
      <CLASS id="new_package">
            <INTERNAL>
                  <OBJ_ID id="package_uid" creation="mandatory"/>
                  <OBJ_ID id="hub_uid" creation="mandatory"/>
                  <TIMESTAMP id="time" creation="mandatory"/>
                  <DOUBLE id="volume" creation="mandatory"/>
                  <DOUBLE id="weight" creation="mandatory"/>
                  <DOUBLE id="value" creation="mandatory"/>
```



```
                <DOUBLE id="tariff" creation="mandatory"/>
                <LONG id="QOS" creation="mandatory"/>
                <DOUBLE id="insurance" creation="optional"/>
                <STRING id="destination" creation="mandatory"/>
            </INTERNAL>
            <EXTENDED/>
        </CLASS>
</WL_DDL>
```

4.3.2.2 Object Class: package_position

Logic: the package_position objects will be used to encapsulate the information needed to notify the enterprise of a package entering or leaving a hub. In the case of the role COLLECTION POINT, this message will be used to notify the enterprise that a package has been collected by the local vector (the package leaves the collection points).
Class Name: package_position
Data Layout:

Property Name	Type	Mandatory	Remark
package_uid	Unique Identifier	YES	the uid assigned to the package by the wrapping process
hub_uid	Unique Identifier	YES	the uid of the hub. In this case this is the unique identifier of the collection point
time	Date/Time	YES	the time when the package is collected from the collection point by the local vector
direction	Integer	YES	0 for inbound, 1 for outbound. The collection point will always set direction to 1

Persistence: we assume this class of objects will only be used to support the interactions between the Collection Point (more in general a hub) and the Enterprise. For this reason we will not give persistence to such class. Even in this case, though, we might want to apply a different logic. For instance we might decide to give persistence to the whole class so that any object of the class package_position presented by the hubs will be automatically stored into the persistence providers managed by WL BOSS. In this case the operation initiated by the CUSTOMER (retrieve information) could be simply represented as a query implementing the logic: "show me all the objects of class package_position that reference (via the package_uid property) the package unique identifier my_package_id and present them ordered by time".

Role Association: COLLECTION POINT (the collection point can interact with the enterprise presenting to WL BOSS an object (event) that belongs to the class package_position).
WL_DDL representation:

```
<WL_DDL>
      <CLASS id="package_position">
            <INTERNAL>
                  <OBJ_ID id="package_uid" creation="mandatory"/>
                  <OBJ_ID id="hub_uid" creation="mandatory"/>
                  <TIMESTAMP id="time" creation="mandatory"/>
                  <LONG id="direction" creation="mandatory"/>
            </INTERNAL>
            <EXTENDED/>
      </CLASS>
</WL_DDL>
```


4.3.3 Business Object: Hub Interaction

In our analysis the hubs interact with the carrier by notifying it of any inbound package and of any outbound package. To this purpose the class package_position, already defined, is enough.

Class name: package_position
Role association: HUB

Please note that the class package_position will be associated with at least two different roles. At the business process level, we can apply a different logic depending on the role (COLLECTION POINT or HUB) presenting an object that belongs to the class package_position.

4.3.4 Business Object: Destination Interaction

Logic: as soon as the package is delivered, the DESTINATION has to notify the enterprise that the package has been accepted by the intended 'recipient'.
Class Name: package_acceptance
Data Layout:

Property Name	Type	Mandatory	Remark
package_uid	Unique Identifier	YES	the package unique identifier assigned during the wrapping process
receiver	String	YES	the name of the person receiving the package and accepting the delivery
time	Date/Time	YES	the time when the delivery has been accepted
acceptance	Integer	YES	1 if the delivery has been accepted 0 if the delivery hasn't been accepted

Persistence: we assume this class of objects will only be used to support the interactions between the Destination and the Enterprise. For this reason we will not give persistence to such class. Even in this case we might want to apply a different logic. For instance we might decide to give persistence to the whole class so that any object of the class package_acceptance presented by the Destinations will be automatically stored into the persistence providers managed by WL BOSS. By doing this, the carrier might implement an historical database in order to support any disputation case. In this case, the class package_acceptance would probably have to be extended in order to contain all the information (such as an electronic signature) needed to support the resolution of any disputation case.

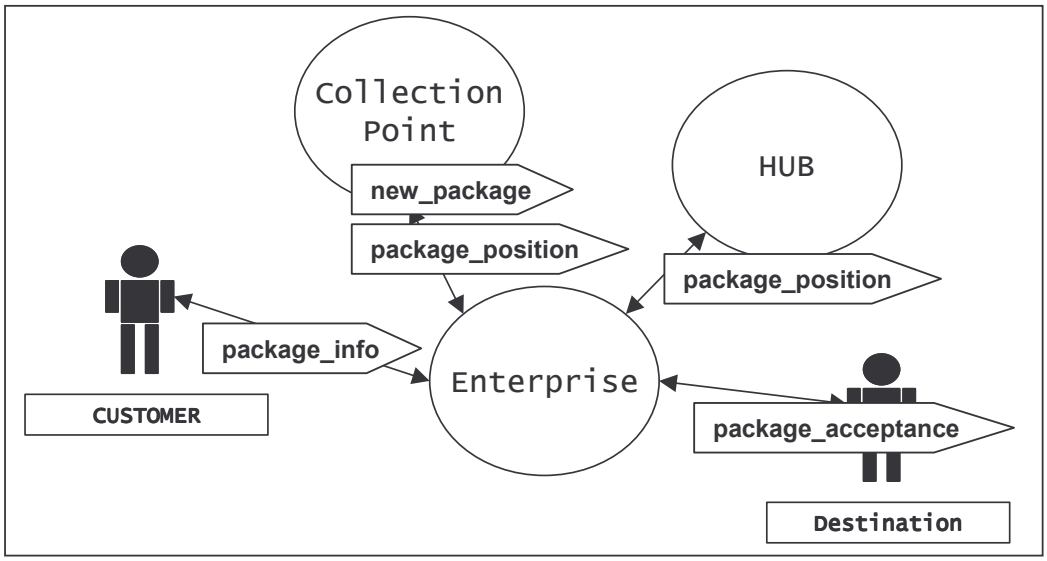
Role Association: DESTINATION (the destination interacts with the enterprise by presenting to WL BOSS an object (event) that belongs to the class package_acceptance).

WL_DDL representation:

```
<WL_DDL>
      <CLASS id=”package_acceptance”>
            <INTERNAL>
                  <OBJ_ID id=”package_uid” creation=”mandatory”/>
                  <STRING id=”receiver” creation=”mandatory”/>
                  <TIMESTAMP id=”time” creation=”mandatory”/>
                  <LONG=”acceptance” creation=”mandatory”/>
            </INTERNAL>
            <EXTENDED/>
      </CLASS>
</WL_DDL>
```


4.4 Stakeholders and Interactions: summary

The picture below shows a summary of all the stakeholders we have identified and of all the ways they can interact with the enterprise.



A customer can ask information about a package, a collection point notifies that a new package has been collected, both the collection point and the hub notify the enterprise that a package has changed its position and last the destination (or recipient) notifies the enterprise that the package has been accepted or rejected.
For each stakeholder we have provided a specific role and for each interaction we have formalized the content (information) that has to be presented to the enterprise. To do so we have used the WareLite Data Definition Language by defining a set of classes that will extend the dictionary of WL BOSS.

Now that we have provided a picture of the stakeholders and of the interactions such stakeholders have with the enterprise, let us have a look at the logic (or business processes) with which the enterprise has to react to such interactions in order to fulfil its business requirements.

4.5 Business Processes

For each identified interaction, presented by a specific stakeholder, we now need to define a business process. A business process is a transactional logic capable of reacting to an event presented to the enterprise by a stakeholder. Event is the technical term to define the information presented to the enterprise whenever an interaction happens. As we have seen, technically an event is an object with a specific data layout as defined by a specific class of objects.

To make this clearer, we will provide an example.
We know that a hub interacts with the enterprise by sending objects belonging to the class package_position to the enterprise. In WL BOSS terms, the action of sending an object to the enterprise is synonym with “presenting an event”. An event provides the representation of something that happens in reality, such as a package that enters or leaves a hub. How does the enterprise have to react to such event in order to support its business requirements?
We know that we are doing all this because the carrier – at any time - must be able to serve the customer by answering the question: “where is my package?”. This means that – at any time - the enterprise must be able to provide at least the last known position of a package. This consideration should immediately make it clear how the enterprise should react to a package_position event: the enterprise must remember the information carried on within the event and it has to be able to provide such information whenever asked.
Our business process, in the simpler case, would react to the package_position event by storing the position information contained into the event. In order to support a more sophisticated logic, the business process might be enriched with other operations such as
a) find the customer information
b) notify the customer that the position of the package has changed
c) if the package position is marked inbound, increment the hub status
d) if the package position is marked outbound, decrement the hub status
e) if the package position is marked outbound, update hub average latency

As you can see a business process can contain logic that immediately supports the services available to the customer (revenue generating services) and logic that is devoted to the improvement/monitoring of operational efficiency (performance monitoring services).
An event is thus the way in which reality is represented, whilst a business process is the way in which an enterprise reacts to something specific that happens in reality.

In the following paragraphs we will describe the business processes that we need in order to fulfil only the major business requirements. We will not go into the details of how to actually code the business processes, instead we will represent such business processes as sequences of operations (or business rules) that can be depicted as self-contained by their nature. Note that such sequences of operations (business processes) can be extended by

adding other operations (i.e. business rules) without affecting the business rules that already populate a business process.

4.5.1 Package_info_BP

The business process Package_Info_BP reacts to an event belonging to the class package_info. In our scenario we have identified only one stakeholder that can present such an event: the customer. Is this really the sole stakeholder that can present such an event? Suppose for a moment that we have another stakeholder, that we will label SYSTEM_ADMIN. Supposedly the information returned to a customer is not as rich (or complex) as the one that is needed by a system administrator. However, both the customer and the system administrator can ask information about an item (package) by presenting an event belonging to the class package_info (that contains the unique identifier of the package they want information about). So, how can you apply two different business processes for these two satkeholders (one returning the information properly formatted for a customer, the other maybe aggregating a lot of information from several distributed systems, as needed by a system administrator) given that both of them will present the same type of event?
WL BOSS' infrastructure can select a business process based on the class of the incoming event and on the role of the stakeholder/agent that presents the event. Assigning different roles to the customer and to the system administrator makes it possible to differentiate the logic required to support these two different types of stakeholders, no matter whether they will present the same type of events.

The relationship that can be declaratively established amongst 1) a class of objects, 2) a role and 3) a business process is called **Ternary Associative Logic**.

Let us go back to the description of the business process that will have to serve a customer enquiring the position of a package: such business process can be described very easily as a sequence of two self-contained operations:

a) load package information
b) reply with package position

In case we want to support the system administrator case, we could immediately form another business process by adding an extra business rule:

a) load package information
b) reply with package position
c) reply with package position history
d) load last-hub (from position history)
e) reply with-last hub information

4.5.2 New_Package_BP

This business process will have to create the persistent data used to represent a package or, more in general, all the data needed to represent the business transaction that is going to happen between the carrier and the customer. The data created by this business process could become the target

for all other business processes whenever e.g. the position of a package changes. For the time being, let us describe this business process with the following very simple sequence of business rules:

a) create package information
b) reply confirmation

The same business process can be extended in order to provide support for other business functions. For instance the same business process can be extended with rules devoted to general ledger management and rules that support planning, for instance related with all the logistics needed to collect packages from the collection point.
An example of a bit more sophisticated business process could be:

a) create package information
b) reply confirmation
c) update general ledger
d) update collection point status
e) notify vector for collection (based on specific CP status)

4.5.3 Package_Position_BP

We have already discussed this business process. In this paragraph we will provide the simpler version and the 'advanced version'. For the simpler version, the business process can be described as having one single business rule:

a) update package position

For the advanced version we can describe more business rules, as shown in the following sequence:

a) update package position
b) find customer information
c) notify customer that the position of the package has changed
d) update hub status
e) update hub average latency

The first rule is quite self-explicative. The second and third rules are devoted to support asynchronous notification to the customer (e.g. based on a set of advanced services). The last rules are devoted to support the monitoring of a hub. Such rule should allow to answer to questions such as: How busy is such hub? How many items have transited such hub in a defined time window? How long does an item stay in such hub (average)?

4.5.4 Package_Acceptance_BP

Which are the operations that have to be initiated by the enterprise when it is notified that a package has reached its final destination? How is the event of a rejected package managed? Here we will describe a basic generic business process that can be extended and modified in order to fulfil more specific business logics:

a) update package position
b) find customer information
c) notify customer that the position has changed
d) if rejected, notify customer
e) if rejected, notify CRM to take action

f) update general ledger
g) mark position information for pending deletion

The very important thing that must be noticed here is that some of the business rules used for this business process might already be available. In fact some of the business rules within this business process have been used in other business processes.

5 IMPLEMENTATION

We have analysed a business landscape and, based on a set of business requirements, we have drawn a first system design. Whilst we have shown some implementation details about how to declare a class of objects within WL BOSS we haven't dug into the details of the technical implementation of a business rule and of a business process. It is not the purpose of this document to expose such details, nonetheless we think that it might be helpful to provide some information about the tools and languages involved in the development of business process and some information about the deployment of the logic on one or more WL BOSS domains, made of Node Managers, the process execution units within WL BOSS.

We have said that a business process is a sequence of business rules, and that a business process will be executed by WL BOSS automatically, depending on the relationships established amongst a class of objects, the role of the agent presenting an event and a business process.

5.1 What is a business rule?

A business rule is a C/C++ body distributed within a DLL. A business rule can be developed with mainstream tools such as Microsoft Visual Studio. Using the same development environment it is possible to test and debug any business rule. Also, WareLite SDK contains a set of tools that facilitate software testing activities. For instance, without having to develop any specialized software agent, it is possible to generate all the workload necessary to test a set of business process both from a logic and from a functional point of view. A business rule is distributed within a DLL (business rule repository). When developing a business rule the developer can assume that the rule will be executed in a single threaded environment. Actually, the same business rule will be executed by a multitude of multi-threaded executives (the Node Managers). Scalability and determinism (i.e. global resources lock management) are not responsibility of the developer but built-in services of WL BOSS.

5.2 What is a business process?

A business process is a sequence of business rules. The business rules referenced within a business processes can be implemented in one or more DLL (business rules repository). Different business processes can use the same sub-set of business rules. A complete business process is the result of the Ternary Associative Logic, associating a role, an event and a sequence of business rules. Whilst the business rules have to be implemented using C/C++, a business process is an execution unit that requires just a configuration activity.
The configuration of a specific business process will thus be the answer to the question 'which are the rules that must be executed within the business process triggered by an Event A presented by Role X? '

