



# FInest – Future Internet enabled optimisation of transport and logistics networks



## D6.2

### Conceptual Design of Event Processing Component

Project Acronym	FInest	
Project Title	Future Internet enabled optimisation of transport and logistics networks	
Project Number	285598	
Workpackage	#WP6 Proactive Event-Driven Monitoring	
Lead Beneficiary	IBM	
Editor(s)	Yagil Engel	IBM
Contributors	Guy Sharon	IBM
	Fabiana Fournier	IBM
	Åsmund Tjora	MRTK
	Michael Zahlmann	KN
	Tor Thunem Knutsen	ARH
	Evert-Jan Van Harten	AFKL
	Reviewer	Metin Turkey
Reviewer	Clarissa Marquezan	UDE

Dissemination Level	PU
Contractual Delivery Date	31.03.2011
Actual Delivery Date	
Version	

## Abstract

*This report presents the second Deliverable of work package WP6 “Proactive Event Driven Monitoring”. Work Package 6 is responsible for the Event Processing Module (EPM), one of the four core technical components of FInest. The role of this module within FInest is to provide event-driven monitoring across domain and transport modality, providing, for instance, FInest’s Business Collaboration Module (BCM) with the runtime information required to achieve three goals: end-to-end visibility of logistics processes, monitoring the achievement of contract terms, and triggering replanning of the scenario when needed.*

*The document describes the second stage of results of the “requirements analysis and selection of technology baseline for the event processing component” (T6.1). This includes a refinement of the results presented in D6.1 in the area of identifying the relevant business requirements provided by domain experts, mainly through the use of functionality demonstrators. The document details the main demonstrator used for extracting EPM requirements: “Real-Time Booking Update Cycle”.*

*The second part of this deliverable is a conceptual design for the Event Processing Module (T6.2) done on the basis of the initial conceptual design described in D6.1, and on the basis of the refined requirements.*

*The document does not describe additional work done in this deliverable: the refinement of generic enablers, which is part of T6.3, “Technological Alignment with FI PPP Core Platform”. The actual results of this work are provided at D9.2 along with the rest of the project.*

## Document History

<b>Version</b>	<b>Date</b>	<b>Comments</b>
V0.1	25/2/2012	Initial internal domain experts review
V0.2	05/03/2012	Second draft, review by internal ICT
V0.3	13/03/2012	All comments incorporated
V0.4	18/03/2012	After additional comments from domain partners
V0.5	25/03/2012	Additional comments from domain+ICT partners
V0.6	27/03/2012	Incorporated changes after official internal review

**Table of Contents**

Abstract ..... 3

Table of Contents ..... 5

List of Tables..... 6

List of Figures ..... 6

Acronyms..... 7

**1. Introduction..... 7**

**2. Finest Demonstrator: Real-time Booking Update Cycle ..... 8**

    2.1. Motivation ..... 8

    2.2. General Information ..... 9

    2.3. Detailed Description ..... 10

**3. Refinement of Requirements..... 14**

    3.1. Event Processing Network Lifetime ..... 14

    3.2. Specific Events to Handle..... 14

**4. Refinement of Design ..... 15**

    4.1. Conceptual Design ..... 16

        4.1.1. Events and Event Sources ..... 16

        4.1.2. Rules and Rules Definition ..... 17

        4.1.3. Detected Situations ..... 17

        4.1.4. Runtime Engine..... 18

    4.2. Novelty in Comparison to Standard Event-Processing Applications ..... 21

    4.3. Interface with Other Finest Modules..... 22

**5. Conclusions and Next Steps ..... 22**

**References ..... 23**

**List of Tables**

Table 1: Additional EPA requirements ..... 14

Table 2: Additional event requirements ..... 15

**List of Figures**

Figure 1: Event Processing Module Conceptual Architecture..... 16

Figure 2: Structure and flow of an example EPN. .... 18

Figure 3: Hierarchy of event processing agent types..... 18

Figure 4: Conceptual architecture of a pattern matching EPA. .... 20

## Acronyms

Acronym	Explanation
EPM	Event Processing Module
BCM	Business Collaboration Module
KPI	Key Performance Indicator
AIS	Automatic Identification System
TPM	Transport Planning Module
TEP	Transport Execution Plan
EPA	Event Processing Agent
CEP	Complex Event Processing
EPN	Event Processing Network

## 1. Introduction

The Event Processing Module (EPM) is a core technical module of the Finest project. Its role in Finest architecture is to collect events from various sources, and perform what is known as *complex event processing (CEP)* operations on them in order to detect situations on which to notify the business collaboration module (BCM) or the frontend of Finest. The availability of such automated response technology will provide Finest users with end-to-end visibility and monitoring of the process, and with the ability to quickly respond to various changes and deviations that often occur in the dynamic and complex logistic processes that Finest is expected to handle. The EPM relies on existing CEP technology; however, as described later, the particular setting of this domain and the specific demands of Finest requires extending the basic event processing architecture towards novel directions.

Finest is developed in a collaborative process between its Domain experts and ICT partners; the idea is to “pull” the requirements from the domain and allow this to lead the technology, rather than to develop technology and “push” it into the domain. The main driver used for this process, during the M12 milestone development, was the *Finest demonstrators*. These demonstrators show specific pieces of functionality; they were developed mutually by domain and ICT partners. The benefit is twofold: first, the demonstrators provide basis for functionality that can be provided by Finest. Second, it facilitates knowledge transfer from domain partners to ICT partners, leading to additional requirements and refinement of the conceptual design of the technical modules.

This document describes the results of this process from the EPM perspective: first, we describe the motivation and functionality of the demonstrator developed in WP6, along with screenshots and detailed explanation (Section 2). Next, we provide the additional requirements that were learned in the development process of the demonstrators (Section 3). Finally, we describe the refined conceptual design of the EPM that evolved during the work on this milestone (Section 4).

## 2. Finest Demonstrator: Real-time Booking Update Cycle

### 2.1. Motivation

Efficient consolidation and asset utilization are major performance measurements for forwarders and carriers. Efficient consolidation and increased asset utilization can improve overall efficiency of the logistic chain, resulting in improved revenues both for forwarders and carriers, and in reduced carbon footprint.

The business motivation behind this demonstrator can be summarized as follows: the goal is to reduce the gap between booking and actual; the accomplishment of this goal would allow forwarders and carriers to better know what freight needs to be transported, allow shippers to be more certain of the space reserved to them by forwarders and carriers, and provide better visibility and predictability on the process they run; overall, much better quality of data. With the improved quality of data, the process will have less disturbances, decreased throughput times, and perhaps, most importantly, a better consolidation for the forwarder and increase of the asset utilization by the carriers.

A root cause analysis for the gap between booking and actual was done in WP2. One of the main causes for the gap is found to be the following: *shippers, carriers, and forwarders may not have the same data about a single booking*. Some of the possible reasons for this can be:

1. Changes often occur in the delivery specification (volume, weight, number of pieces), and often at the last moment (e.g., initial measurements are not accurate, or items are repacked). Shippers sometimes do not inform on time about changes in the booking, for various reasons; for example:
  - a. There is no structured process to do that (or, as identified in root cause analysis: shippers are not asked to update changes in their shipping volume or weight when pickup time approaches).
  - b. Busy operational schedule (hence shippers either forget to update, or do not see this as high priority).
2. Human or technical errors in transferring data between different applications.

The demonstrator shows a solution to prevent the root causes mentioned above. It provides a single cloud location for the order, with the same details available to all parties at the same time, and supports structured information exchange that will allow smooth and collaborative changing of a booking.

There could be many other reasons for the gap, such as risk management of shippers, wrong measurements, and last minute changes. Another part of the demonstrator scenario addresses these cases: we reached the pickup time of the shipment, and the truck driver discovers a discrepancy between the booked and actual shipment. In that case the goal is to reduce the time gap in propagation of capacity information: the earlier all stakeholders in the chain are aware of the change, the better they can adapt. Examples of solutions are: change timing of the transport plan (and potentially postpone pickup), find other carriers, or change priority of different customers in the consolidation of a forwarder.



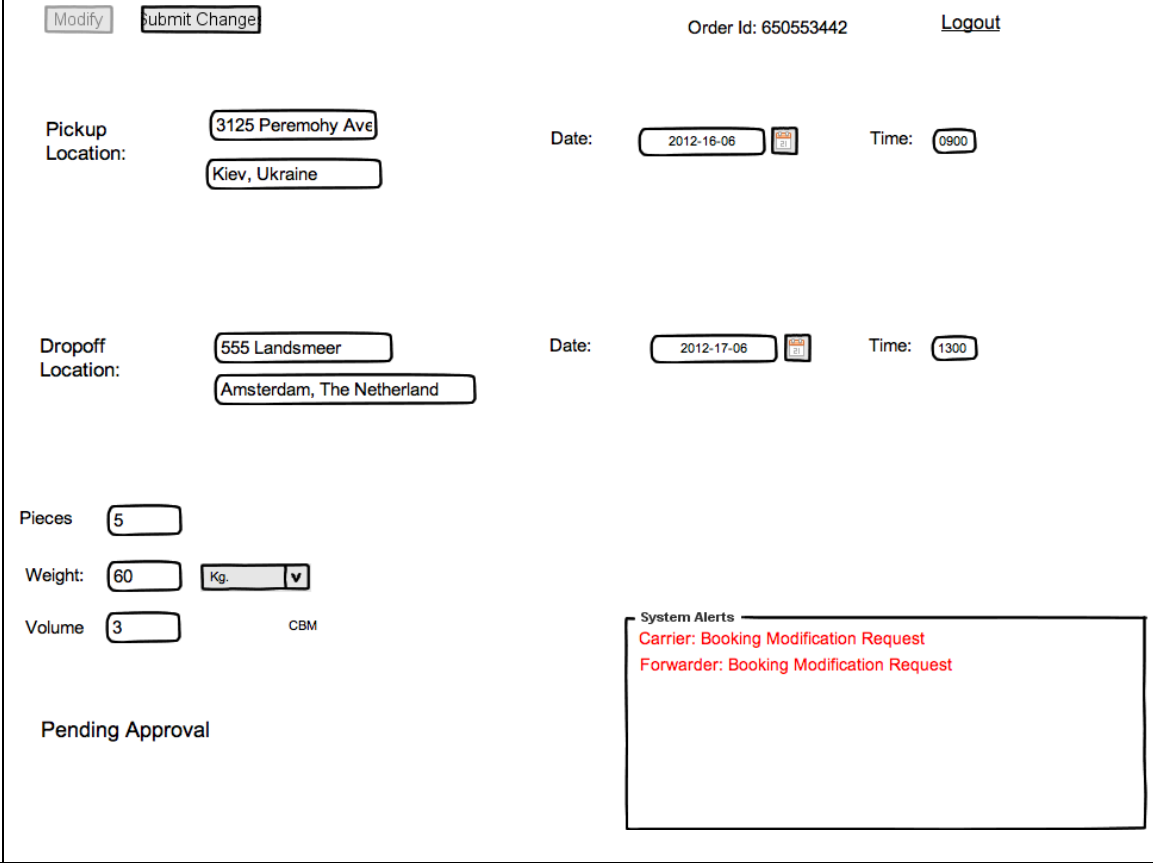
## 2.2. General Information

<i>General Information</i>	
<i>Element</i>	<i>Description</i>
<i>Title</i>	Real-time booking update cycle
<i>As-is scenario</i>	<ul style="list-style-type: none"> <li>• For various reasons, shippers, carriers, and forwarders see different information regarding the same booking.</li> <li>• Changes in booking are either done by shipper without notice (shippers simply present the pickup driver with different sizes / weights / no. of pieces than those booked), or propagated to one stakeholder in the chain (only a forwarder is notified but the carrier is not updated immediately)</li> <li>• No online collaborative process exists to fix booking discrepancy.</li> </ul>
<i>Addressed main challenge</i>	Discrepancies between booked shipment and actual shipment are a major source of shipment delays and under or over utilization of resources.
<i>To-be scenario</i>	<ul style="list-style-type: none"> <li>• “The single supply-chain truth”: a single order record, which can be accessed by all the partners, according to their roles, as long as agreed by the data owner.</li> <li>• Complete booking modification cycle: shipper can submit modification to the “single truth”, this alerts all relevant parties that try to accommodate the request and produce new price quote; finally, shipper can agree, or not, to the new price.</li> <li>• Online collaborative handling of booking discrepancy: when discrepancy discovered at pickup time, another cycle of alert – replan – new price quote – confirmation of shipper is performed.</li> </ul>
<i>Demonstrator approach</i>	<p>The demonstrator shows the views of shipper, forwarder, carriers, and truck drivers. First, they all have access to the same reservation that is stored in the Finest cloud. Then, we show a full booking modification cycle: the shipper submits modification (in any of the booking parameters), which alerts the other players. The carriers and the forwarder use replanning module (not part of this demonstrator) to see if the modification can be accommodated; the combined result is sent back to the shipper who needs to accept the new price quote.</p> <p>Finally, a similar modification cycle can also occur during execution, when it is <u>triggered by a truck driver who discovers a booking discrepancy at pick-up</u></p>

time.	
<i>Involved modules</i>	EPM, BCM, (TPM – implicitly)
<i>Lead</i>	IBM
<i>Involved partners</i>	KN, AFKL, NCL

### 2.3. Detailed Description

<i>Detailed Information</i>	
<i>Element</i>	<i>Description</i>
<i>Demonstrator title</i>	Real-time booking update cycle
<i>Screen name</i>	Shipper: booking modification

<p><i>Screenshot</i></p>	
<p><i>Screen purpose</i></p>	<p>The shipper views the current order, the same “truth” as viewed by the other parties, and can submit booking modification.</p>
<p><i>Detailed description</i></p>	<p>The screen allows the shipper to request a modification of any of the parameters. After entering new values, the shipper clicks “submit changes”. At this point the modification is shown to be “in process”. At the end of the cycle, when the carrier and forwarder returned with a new price quote, the shipper sees the new quote and has “confirm” and “reject” buttons.</p> <p>Note that the “System Alerts” box is not shown in actual application. In the demonstrator it shows that when the shipper submits the request, both the relevant carriers forwarder are alerted that such a request has been submitted.</p>
<p><b><i>Detailed Information</i></b></p>	
<p><i>Element</i></p>	<p><i>Description</i></p>
<p><i>Demonstrator title</i></p>	<p>Real-time booking update cycle</p>
<p><i>Screen name</i></p>	<p>Truck Driver: report booking discrepancy</p>

<p><i>Screenshot</i></p>	<div style="text-align: right;"> <span>Order Id: 650553442</span>      <a href="#">Logout</a> </div> <div style="margin-top: 20px;"> <span>Modify</span>    <span>Submit Changes</span> </div> <div style="margin-top: 20px;"> <p>Pickup Location: <input type="text" value="3125 Peremohy Ave."/>  <input type="text" value="Kiev, Ukraine"/></p> <p>Date: <input type="text" value="2012-10-06"/>       Time: <input type="text" value="0900"/></p> </div> <div style="margin-top: 20px;"> <p>Pieces: <input type="text" value="5"/></p> <p>Weight: <input type="text" value="65"/> <input type="text" value="Kg."/> <input type="text" value="v"/></p> <p>Volume: <input type="text" value="3"/> <input type="text" value="Meters"/> <input type="text" value="v"/></p> </div> <div style="margin-top: 20px;"> <p>Pending Approval</p> </div> <div style="border: 1px solid black; padding: 5px; margin-top: 20px;"> <p><b>System Alerts</b></p> <p>Carrier: Booking Modification Request</p> <p>Forwarder: Booking Modification Request</p> </div>
<p><i>Screen purpose</i></p>	<p>The truck driver views the current order, reports it, and informs in what way the actual shipment is different than the specification.</p>
<p><i>Detailed description</i></p>	<p>The screen allows the driver to report new values in the shipping details (weight, size, number of pieces) and submit the changes. At this point the modification is shown to be “in process”. At the end of the cycle, when the carrier and forwarder returned with a new price quote, and the shipper confirmed the new price, the driver sees that he can proceed with the pickup.</p> <p>Once the truck driver submits the notification, both the carrier and the forwarder are immediately alerted and can access the system to view the discrepancy.</p>

<i>Detailed Information</i>	
<i>Element</i>	<i>Description</i>
<i>Demonstrator title</i>	Real-time booking update cycle

<i>Screen name</i>	Carrier / forwarder: process booking modification
<i>Screenshot</i>	<div style="text-align: right;">Order Id: 650553442 <a href="#">Logout</a></div> <p>Pickup Location: 3125 Peremohy Ave. Kiev, Ukraine</p> <p>Date: 2012-16-06 Time: 0900</p> <p>Dropoff Location: 555 Landsmeer Amsterdam, The Netherland</p> <p>Date: 2012-17-06 Time: 1300</p> <p>Pieces 5 Carrier Charges: 2475 EUR</p> <p>Weight: 60 confirmed:50 Kg.</p> <p>Volume 3 CBM</p> <p><b>REPLAN</b> SUCCESS recommended charge: 2775</p> <p>Revised Charges: <input type="text" value="2700"/> EUR</p> <p>Pending Shipper Approval <input type="button" value="Approve"/></p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p><b>System Alerts</b></p> <ul style="list-style-type: none"> <li>Carrier: Booking Modification Request</li> <li>Forwarder: Booking Modification Request</li> <li>Forwarder: Carrier Response to Shipper Modification Request</li> </ul> </div>
<i>Screen purpose</i>	The carrier or forwarder views the current order, and processes a booking modification request submitted by the shipper.
<i>Detailed description</i>	<p>The carrier or forwarder views the current order, the same “truth” as viewed by the other parties.</p> <p>The following flow is triggered when either a booking modification request is submitted or booking discrepancy is reported:</p> <p>The modifications are show in red next to the confirmed (previous) values.</p> <p>The carrier / forwarder can click the button “replan” that opens the replanning module (beyond the scope of this demonstrator).</p> <p>The result from the replanning module is either “success” with new recommended price, or “fail” (in the demonstrator, always success).</p> <p>The carrier / forwarder can modify the price, and click “send”.</p> <p>The status is shown to be “waiting for customer confirmation”. After the shipper confirms, this message is removed and the new values are shown as part of the current reservation.</p>

### 3. Refinement of Requirements

Based on the interaction with domain partners, through the development and usage of the demonstrators, we define sets of more subtle requirements, in addition to those stated in D6.1.

#### 3.1. Event Processing Network Lifetime

The new requirements below were not mentioned explicitly in D6.1; these are requirements that are not usually obtained by current event processing systems and are unique to the domain of monitor logistic executions.

The lifetime of a shipment execution is strongly related to the notion of *Transport Execution Plan (TEP)*, which is adapted from FP7 project efreight [8]. The TEP encapsulates the information regarding the execution of a single shipment, including the event processing rules that should be instantiated to monitor that shipment. The notion of TEP is described in detail in D7.2.

The two requirements R\*101 and R\*102 ensure the ability of the system to dynamically support TEPs that are created while the system is up and running. R\*102 and R\*103 allow the system to be changed dynamically when more sources of data are available and more domain requirements surface; they allow the configuration of the system without the traditional distinction between “design time” and “run time”.

**Table 1: Additional EPA and rule engine requirements**

R*101	Plan Instantiation	Ability to instantiate event processing unit (EPA) as a result of Transport Execution Plan (TEP) requirement, and in the context of plan id, and parameterize it according to the TEP.
R*102	EPA Termination	Ability to terminate an EPA when plan execution completes.
R*103	Addition of Events	System operators can (easily) add events (not known in design time) that the system should monitor.
R*104	Addition of Rules	System operators can add additional rule templates (see requirement R201 from D6.1), not known in design time.

#### 3.2. Specific Events to Handle

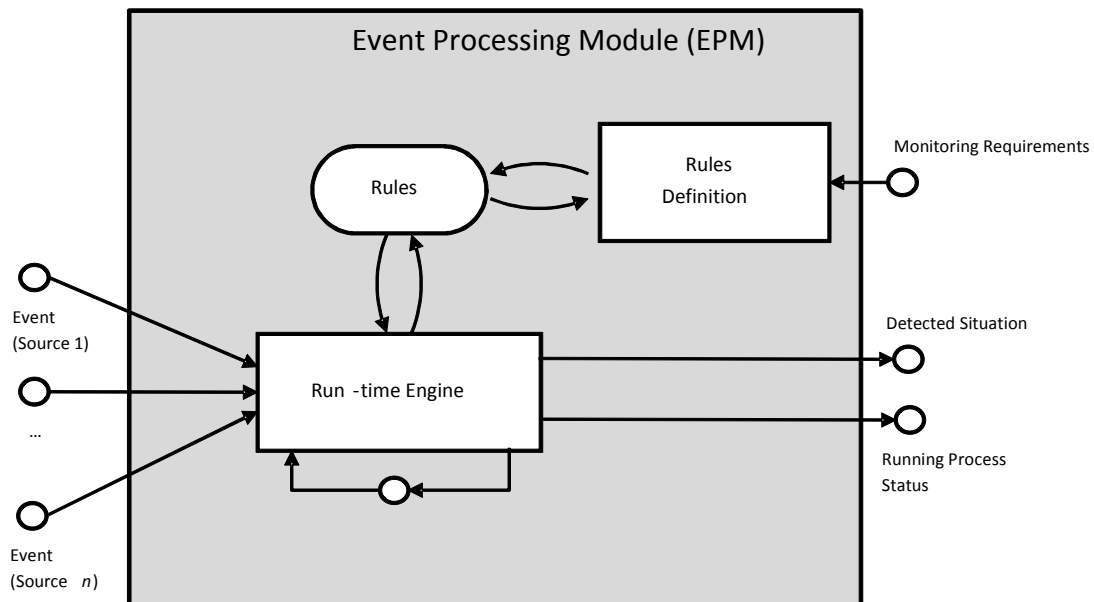
A preliminary list of events was described in D6.1. The following are types of events that were gathered through the use of the demonstrators, and were either not mentioned in D6.1, or mentioned at a higher level. The table indicates the *source* of the event, and the target for *notification*, if exists.

**Table 2: Additional event requirements**

<b>Id</b>	<b>Event</b>	<b>Description</b>	<b>Source</b>	<b>Notification</b>
R*201	Port Call Request	Port call request event, instantiate port call context.	Ship	BCM
R*202	Port Service Request and Confirmation	Port service request (for specific port resource: pilot, quay, tugboat, mooring crew, ground equipment, cargo workers, electric and water supply, storage space)	Ship	BCM Comment: Handle confirmation event submitted to system and notify BCM as well
R*203	Booking Cancellation		Shipper, through Finest system	Carriers, Forwarder
R*204	Booking Modification		Shipper, through Finest system	Carriers, Forwarder
R*205	Booking Modification Approval	Approval of booking modification, possibly with new price quote	Carrier or Forwarder, through Finest system	Shipper
R*206	Price Change Approval	Approval of new price quote, resulting from a booking modification.	Shipper, through Finest system	Carriers, Forwarder

## 4. Refinement of Design

This section provides a conceptual design of the EPM. A high-level conceptual design was already presented in D6.1; we repeat the essence of the design here for completeness purposes, and further elaborate on it. We then elaborate on the interaction with other Finest modules.



**Figure 1: Event Processing Module Conceptual Architecture**

## 4.1. Conceptual Design

The Event Processing Module (EPM) includes the following components, which are identified in Figure 1:

- Events and Events Sources
- Rules and Rules Definition
- Runtime Engine
- Detected Situations

### 4.1.1. Events and Event Sources

Finest EPM will monitor various events that are relevant to the execution of a transport plan, from the time a booking is established. Most event sources and the means of communication with them will be defined in the application, preferably through a FI-Ware generic enabler.

Some of the events are *internal*; for example, as shown in the demonstrator in this deliverable, a truck driver may issue a pickup event, or alternatively, submit a booking discrepancy event prior to loading the shipment. Other events are *external*; they are received from various electronic systems such as AIS, Airport, and existing tracking systems employed by freight forwarders. In both cases, we call the entity that produces the event an *event producer*. The set of events handled by the system will be accessible directly by a user interface, to support Requirement R\*103.



### 4.1.2. Rules and Rules Definition

The behaviour of the EPM is defined by event processing rules. The two parts of an event-processing rule are:

1. Event Pattern (the rule *body*): may include a single event, or an operator such as filter, join, aggregation, or trend, defined over a set of event types, along with mathematical conditions. For example:
  - a. A single event, which is a temperature read from RFID, with conditions comparing its value with the allowed range.
  - b. A trend operator over a series of temperature reads, identifying steady increase.
  - c. Aggregation operator: sum the gap between actual cargo and expected cargo of flights departing towards a specific airport; a mathematical operation can then forecast the resulting cargo backlog at that airport.
2. Derived Event (the rule *head*): if the pattern in the rule body is matched, the rule triggers, and the result is that an event (which is a “complex event”, or “high-level event”, representing the occurrence of the pattern) is created and emitted. The resulting derived event can be sent to a consumer (for example, notify a user) or serve as an input event to another rule in a chain.

The transport and logistic application is a system that monitors a large number of short-term executions (shipments), both concurrent and sequential. The solution in Finest is a *configurable rules system*, with a structure of three layers:

1. Generic event-processing platform, as described above.
2. A set of parameterized rules (that can be edited and maintained by designer); these are called *rule templates*. For example: if neither event “pickup” nor event “booking modification” are received within X min of scheduled pickup time, change status to “delayed pickup”. Here, X is a parameter of the rule (in general, multiple parameters are possible). The set of rules will be accessed directly from user interface, to support Requirement R\*104.
3. Specific logistic scenarios: For each particular logistics scenario the following two configurations are being done: (i) which rules are activated for this scenario, and (ii) the value of the parameters to each rule. In the example above, for a specific TEP we might ask this rule to be instantiated, and provide the parameter “30 minutes”.

This architecture is a significant departure from our chosen technology baseline AMiT (see [2] and D6.1). The parameterized set of rules will be created through interaction with domain experts, mainly through the use of demonstrators. An initial step in this process was the extraction of the set of events presented in the previous section.

### 4.1.3. Detected Situations

A triggering rule represents the detection of situation; this is communicated to the rest of the system, as mentioned above, through an event or a derived event. A situation detected by the event processing engine is reported to *event consumers*, which in the case of Finest refer to the external modules that receive events from the EPM; this is discussed in Section 4.2.

#### 4.1.4. Runtime Engine

The runtime engine is the piece of software that is responsible to detect during runtime which rules are matched. The engine of Finest will be designed according to the generic *event processing network (EPN)* architecture summarized by [1] and [5]. The EPN (as illustrated in Figure 2), is the conceptual view of the flow of events to, within, and from the runtime engine. The EPN consists of event producers and consumers (described above), and the runtime engine with event processing agents and channels (discussed below).

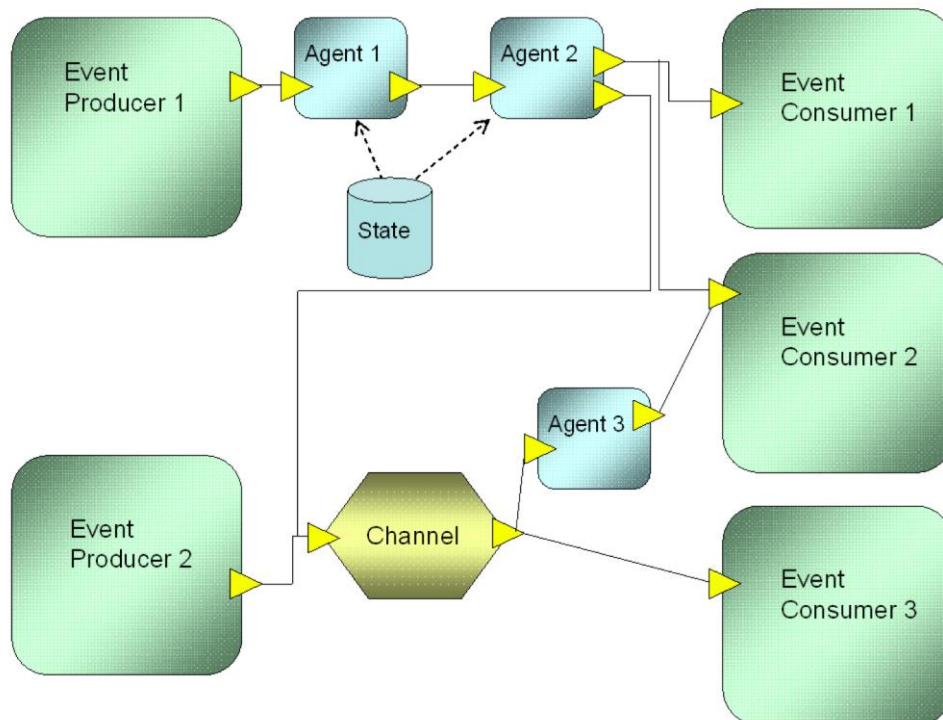


Figure 2: Structure and flow of an example EPN.

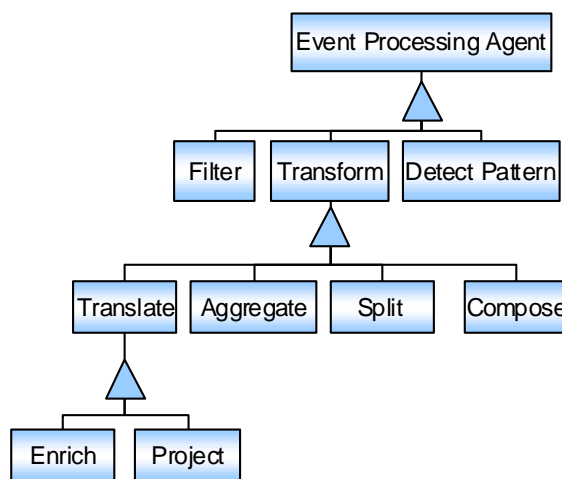
##### 4.1.4.1. Event Processing Agents

Event processing agents (EPA) are local processing units with a well defined logical role. In Finest, an EPA corresponds to a rule in the rule base. Each EPA is responsible for the detection of the event pattern in the rule body, and emits derived events as a result according to the rule head. Because rules are selected and instantiated per execution, the EPAs will be created in runtime, according to the set of rules selected.

An EPA has several types, depending on the operation it is expected to perform:

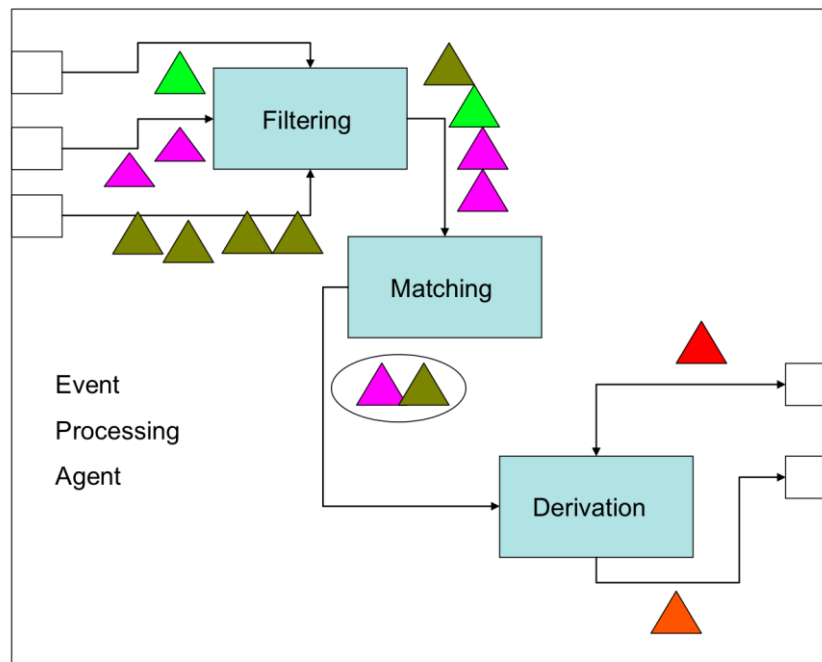
- *Filter* determines whether an input event continues its way on the flow
- *Transform* creates derived events based on function on the content of the input events, as seen in the figure there are several sub-types:
  - *Translate* (1:1) – emits a single derived event as a function of a single input event, this may include enrichment from external data source, or projection
  - *Aggregate* (N:1) – emits a single derived event as some aggregate over multiple input events

- *Split* (1:N) – emits multiple events using multiple derivation functions over a single input events
- *Compose* (M:N) – similar to SQL join with a variety of options.
- *Detect pattern* – emits zero or more derived events as a result of pattern matches among possibly different types of events (example: the pattern sequence (E1, E2, E3) is matched when there are three events of types E1, E2, E3 that occur within the specified order).



**Figure 3: Hierarchy of event processing agent types**

The hierarchy of event processing agent types is illustrated in Figure 3. The three high-level functionalities described above can be distributed between different agent types with specific purpose. However, it is possible and sometimes more convenient to generalize the concept of EPA to perform all: filtering, then pattern matching, and then transformation. This is illustrated in Figure 4.



**Figure 4: Conceptual architecture of a pattern matching EPA.**

For Finest, *Filtering* can be applied to isolate those events that may affect a specific shipment; for example, filter traffic reports from the specific highway numbers that are relevant to the shipment. *Transform* can, for example, *aggregate* the changes of bookings for a forwarder in order to optimize their consolidation.

*Pattern detections* are required for most operations. An event pattern consists of three parts:

1. Set of event types
2. Constraints over attributes of events, which effectively filter the set of tuples of events that are used in the pattern matching.
3. Operator over the event types: indicates the kind of join operations we look for. Below we provide a list of the most common high level patterns used in CEP and that will be used in Finest.

The patterns operators that will be implemented for Finest are as follows:

1. **All**: triggers if *all* of the event types appear within a specific time window.
2. **Any**: triggers if *any* of the types appear within a specific window.
3. **Absence**: triggers if *none* of the event types appear within a specific window.
4. **Sequence**: triggers if all of the event types appear within a specific time window in a specific order.
5. **Trends**: a pattern that indicates a specific change over time of some observed value; this refers to the value of a specific attribute or attributes.

#### 4.1.4.2. Channels

In order to allow flexible routing of events from event producers to EPAs, between EPAs, and from EPAs to consumers, we employ *event channels*; an entity whose role is to collect and dis-

tribute events between objects in the system. A channel can be implemented as a multicast protocol, or other message oriented middleware. The main idea is to avoid having each source (producer or EPA) connect to each event destination (EPA or consumer), but have all of them connect to a channel, which contains the routing logic.

A *routing scheme* denotes the type of information used by a channel to make a routing decision. Here is a short description of each of these routing schemes:

- *Fixed*—The channel routes every event that it receives on any input terminal to every output terminal. In cases where there are multiple output terminals this means that separate copies of each input event are transmitted on each output terminal.
- *Type-based*—The channel makes routing decisions based on the event type of the event that is being routed.
- *Content-based*—The routing decision is based on the event's content. This can be phrased as decision trees or decision tables, and is based on the input event content, and possibly also on context information.

#### 4.1.4.3. Context

A context is a specification of conditions that group event instances so that they can be processed in a related way. Any EPA is associated with specific context or contexts; each context defines the sets of events that will be processed by that EPA as a group. There are several types of context, as identified by [1] and [3]:

1. Temporal: the most common type of context; it defines a time window, so that all events that fall in the same time window are processed together. A temporal context window is often *sliding*; meaning that the current context window of size X minutes includes the events in the last X minutes.
2. Spatial: grouping of events according to their geospatial characteristic; this can be done either with explicit coordinate system, or using named locations such as cities or airports.
3. Segmentation: grouping of events according to some attribute of events; for example, a segmentation context can be defined to group events that have the same shipment id.

Note that through the concept of context, events have the ability to initiate an EPA instance. If an EPA is defined for some context, then each time an event that requires a new context partition is detected, a new EPA is created. For example, with segmentation context, when an event with a new value for the segmented attribute is received, an EPA is created, and all subsequent events with the same value are channeled to that EPA.

## 4.2. Novelty in Comparison to Standard Event-Processing Applications

The Finest project presents interesting challenges to the paradigm of event-processing. Most event driven applications are created on top of the engine and run “forever” (of course with various adaptations once in a while); for example: fraud detection in financial institutions, machinery condition monitoring for heavy industries, and operational management of call centres. The transport and logistic application, in contrast, is a system that monitors a large number of short-term executions (shipments), both concurrent and sequential. This creates two main challenges:

1. There is no one specific rule set that is always valid. Each shipment may use a different subset of the rules. Moreover, the values within rules (for example, specific temperature that the shipment should be kept in, the specific time delay that becomes crucial) are specific to shipments. The solution to this challenge in Finest is the 3-layered configurable rules system, detailed in Section 4.1.2.
2. In standard event processing applications, the event processing network is static, and specific agents can be initiated when monitored objects are created (such as new customer). In Finest, in contrast, a new TEP requires a whole network of EPA to be initiated, supporting a unique set of rules. As formalized by Requirements R\*101 and R\*102, the system should be able to initiate the set of EPAs for a specific TEP when execution starts, and terminate them when the execution ends. In the design of Finest EPM the concept of context is exploited to support this functionality: we use segmentation context, in which segmentation is according to plan Id. We call this *TEP lifespan*: initiate a context (and a set of EPAs) when TEP event with a new plan Id is received, and terminate the context (and thus all associated EPAs) when a TEP conclusion event is received. This constitutes the dynamic creation and termination of the EPA structure per TEP, as required by R\*101 and R\*102.

### 4.3. Interface with Other Finest Modules

The EPM communicates with the Business Collaboration Module (BCM), in two directions, as detailed below. In addition, EPM may send notifications to Finest frontend.

#### **Input:**

*Set of rules to instantiate from the Transport Execution Plan (TEP)*

The TEP provides the configuration discussed under “rules definition” above, by (i) specifying which of the rules and the rules templates should be instantiated to monitor the execution of this plan, and (ii) provides parameters to the relevant rule templates.

#### **Output:**

1. *BCM: events related to shipment current status*

Notifications sent to BCM according to the final set of rules employed by the EPM for a specific TEP. These notifications let the BCM update status of all shipment monitoring parameters, and provide each role the alerts and data required by that role.

2. *Frontend: notification on future events*

Some notifications are not required by the BCM – these are notifications that do not affect the status of any shipment parameter, but rather provide early warning on an event that is likely to affect a shipment in the future [4,7].

## 5. Conclusions and Next Steps

In this document we perform the second step in the design of an event processing module for Finest. We first present the demonstration exercise done to facilitate knowledge transfer from domain partners to ICT partners: a demonstrator was defined in a collaborative process, answering specific business needs identified in the root cause analysis. Next, we generate new re-

quirements, in addition to those covered in D6.1, as a result of the work with this demonstrator and other demonstrators. This process leads to a refinement of the conceptual design (initially presented in D6.1), where we describe the main components of the EPM and the relationships among them, we stress the novelty of our design in comparison to standard event-processing applications, and describe the relationship with other Finest modules.

The next step in this work package is an initial technical design of the EPM. This involves more specific data structures for input and output, more comprehensive lists of events to handle, explicit rule templates, and possibly specific event processing agents and event processing network.

## References

- [1] Opher Etzion and Peter Niblett, *Event Processing in Action*, Manning, 2010.
- [2] Asaf Adi and Opher Etzion, *AMiT - the situation manager*. VLDB J. 13(2): 177-203 (2004)
- [3] Opher Etzion, Yonit Magid, Ella Rabinovich, Inna Skarbovsky, and Nir Zolotorevsky, *Context aware computing and its utilization in event-based systems*. Distributed Event-Based Systems (DEBS), 2010.
- [4] Segev Wasserkrug, Avigdor Gal, and Opher Etzion, *A Model for Reasoning with Uncertain Rules in Event Composition Systems*. Uncertainty in Artificial Intelligence (UAI), 2005.
- [5] Cathrine Moxey et al., *A Conceptual model for Event Processing Systems*, An IBM Redguide publication, <http://www.redbooks.ibm.com/redpapers/pdfs/redp4642.pdf>, 2010.
- [6] <http://www.iata.org/whatwedo/cargo/cargo2000/Pages/master-operating-plan.aspx>
- [7] Yagil Engel and Opher Etzion, *Towards Proactive Event Driven Computing*, in Distributed Event-Based Systems (DEBS), 2011
- [8] <http://www.efreightproject.eu/>