FInest – **F**uture **In**ternet **e**nabled optimi**s**ation of **t**ransport and logistics networks

D6.5

Final technical specification and phase 2 implementation plan for the 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	Fabiana Fournier	IBM
Contributors	Sarit Arcushin	IBM
	Guy Sharon	IBM
Reviewers	Rod Franklin	KN
	Clarissa Marquezan	UDE
Dissemination Level	Public	
Contractual Delivery Date	30/3/2013	
Actual Delivery Date	30/3/2013	
Version	V1.0	

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement no. 285598

Abstract

This report presents the fifth and last deliverable of work package 6 "Proactive Event Driven Monitoring" accountable for the definition and implementation of the event processing module in FInest. It describes the final technical specification of the Event Processing Module in FInest along with the implementation plan of this module in phase 2 of the FI PPP Programme.

The event processing module will be one of the two core modules in the envisioned cSpace follow up project platform. This fact stresses the importance and the generic nature of the event processing module, positioning it at the heart of the envisioned cSpace platform.

Document History

Version	Date	Comments
V0.1	20/11/12	First draft
V0.2	20/1/13	Updates after project technical meetings
V0.3	15/2/13	Updates after the technical face-to-face meeting in Essen
V0.4	1/3/13	Updates after project face-to-face meeting in Hamburg
V0.5	15/3/13	Updates after review
V1.0	30/3/13	Submitted version

Table of Contents

List of Tables

List of Figures

Acronyms

Acronym	Explanation
BCM	Business Collaboration Module
CEP	Complex Event Processing
EPA	Event Processing Agent
EPM	Event Processing Module
EPN	Event Processing Network
GE	Generic Enabler
FInest	Future Internet enabled optimisation of transport and logistics networks
JMS	Java Message Service
JSON	JavaScript Object Notation
IoT	Internet of Things
TCP	Transport Chain Plan
TEP	Transport Execution Plan
TPM	Transport Planning Module
WP	Work Package

1. Introduction

The Event Processing Module (EPM) is one of the four core technical modules in the FInest (Future Internet enabled optimisation of transport and logistics networks) project. Its role is to collect events from various sources and perform complex event processing on them in order to detect situations of interest; that is, of relevant meaning to the consumer of the event enabling them to react or make use of the event appropriately. In FInest, these *detected situations* (aka *derived events*) are notified to the Business Collaboration Module (BCM), or to the frontend of FInest, so appropriate actions can be taken, e.g., triggering re-planning of the transport plan.

This report is comprised of two interrelated parts. The first part is the refinement of the EPM specification delivered in Month 18 of the project, including its interactions with other core modules of FInest, and the potential use of FI-WARE Generic Enablers (GEs). The second part is a plan for implementation of the EPM in phase 2 of the FI PPP Programme. The FInest follow up project, *cSpace* (Future Internet Business Collaboration Networks in Agri-Food, Transport and Logistics), will leverage the work achieved in FInest and extend the FInest platform to the proposed cSpace platform. In fact, the event processing module will become one of the key core modules on which the cSpace platform will rely on. More precisely, the EPM will be reflected in two subtasks in the follow up project: The event management component of the real-time B2B collaboration module (Subtask 242) and the "Real-time exception detection and handling" baseline application (Subtask 451).

With regards to the first part of this report, i.e., refinement of the EPM specification, it should be noted that the specification presented in D6.3 "Initial technical specification of event processing component"[1] has been revised and validated through the prototype implementation of the FISH use case (refer to D6.4 "Prototypical implementation of event processing component" to be submitted at M24 of the project) and was found both correct and valid. As a result, the authors haven't updated the EPM specification presented in D6.3, and included the main parts in this document, for the sake of completeness. For background on event processing in general, and proactive event driven computing in particular, as well as the methodology applied, refer to D6.3[1]. Table 1 shows the relationships between D6.3 and D6.5 sections to assist the reader.

Table 1: Relation between D6.3 and D6.5 EPM specification sections

Section in D6.3	Section in D6.5
2. Background on event processing	Omitted
3. Methodology	Omitted
4. FInest EPM architecture overview	Section 2 - **Slightly** modified to reflect the actual implementation adopted
5. FInest EPM technical specification	Section 3 - Copied

[1] Available at http://www.finest-ppp.eu/

Appendix A - Event processing network for the FISH use case	Refined and implemented in FInest EPM prototype (refer to D6.4 "Prototypical implementation of the event processing component" to be submitted in M24). Omitted in this document.

This document is organized as follows: Section 1 introduces the report scope and main goals. Section 2 provides the EPM architecture overview. Section 3 presents the EPM technical specification. Section 4 details the phase 2 implementation plan of the event processing module. The report concludes with a brief summary.

2. FInest EPM architecture overview

The event processing module in FInest comprises a run-time engine, event producers, and event consumers. Specifically, the CEP (Complex Event processing) engine of FInest includes an integrated run-time platform to develop, deploy, and maintain reactive and proactive applications using a single programming model.

This section gives an overview of the FInest EPM architecture with relation to the CEP GE offered by FI-WARE, including event types and interfaces definitions, and descriptions of the EPM application and instantiation. It is important to note that in essence FInest EPM extends the CEP GE provided by FI-WARE by adding proactive capabilities. Therefore, the "reactive" parts as well as the interfaces described in this section are actually the ones provided by the CEP GE and customized to FInest requirements. We intentionally conserved the same section structure of sections 2 and 3 in this report to its counterparts 4 and 5 in D6.3, so the reader of D6.3 could simply skip sections 2 and 3 in the present document.

2.1. Event types and attributes

Events enter the CEP engine during run-time. They carry information about things that happen externally to the engine (raw events) or as a result of processing in the engine (derived events). An *event* is an object of an *event type* and its *attributes* are defined based on the event type. Events are actual instances of the event types and have specific values. For example, the event "today at 10 PM a customer named John Doe booked a new order" is an instance of the Order event type.

We distinguish between two types of event: raw events and derived events.

1. *Raw event*: A raw event is an event that is introduced into an event processing system by an event producer. The definition of a raw event relates only to its source and not to its structure; a raw event may or may not be composed of other events.
2. *Derived event*: A derived event is an event that is generated as a result of event processing that takes place inside an event processing system. When a derived event is emitted outside the event processing system and consumed by a consumer, it is called a *detected situation*.

Every event instance has a set of built-in attributes (metadata). The EPM employs the following attributes in the event type's metadata:

- *OccurenceTime* – a timestamp attribute, which we expect the event source to fill in as the occurrence time of the event. If left empty, this equals the *detectionTime* attribute value.
- *DetectionTime* – a timestamp attribute that records the time the CEP engine detected the event. The time is measured in milliseconds, specifying the time difference between the current machine time at the moment of event detection and midnight, January 1, 1970 UTC.
- *Duration* – a timestamp attribute that stores the time duration of the event in milliseconds in case the event occurs within a time interval.
- *EventId* – a unique string identification of the event, which can be set by the event source to match the asynchronous output for the event.
- *EventSource* – holds the source of the event (usually the name of event producer).

The above built-in attributes can be used in an expression in the same manner as user-defined attributes. User defined attributes can be added to the event class by defining their types. If the attribute is an array, its dimension should be specified.

When one of the event processing agents detects a situation, it can create one or more derived event instances. These event instances have the same characteristics as an input event; they have both user-defined (event payload) and built-in (event header) attributes. When derived events are emitted to consumers they are called *detected situations.*

2.1.1. Proactive attributes

Future events in the FInest CEP engine are expressed by setting the event a future occurrence time with an optional distribution. Its occurrence certainty can be expressed using the *certainty attribute*. The following attributes are added to the event type built-in attributes:

- *Certainty* – a built-in double attribute that stores the certainty of this event. An event has a default certainty value equal to 1, while it can have any value between 0-1.
- *Cost* – the cost of this future event occurrence. The cost is negative if this is an opportunity. This attribute is used by the proactive algorithm to decide on mitigation actions to prevent the occurrence of a predicted event.
- *ExpirationTime* - the time until which the cost and certainty parameters are valid, and until which an activation of proactive action is considered by the proactive algorithm (beyond this time there is no point in changing any course of action).

2.2. Interfaces

The CEP runtime engine has three main interfaces with its environment as depicted in Figure 1.

1. Input adapters for getting incoming events
2. Output adapters for sending derived events
3. CEP application definition (build time)

The application definitions, i.e. the EPN (Event Processing Network), are written by the application developer during the build-time. The definitions output, in JSON (JavaScript Object Notation) format, is sent to the CEP run-time engine. At run-time, the CEP receives incoming events through the input adapters, it processes these incoming events according to the definitions, and sends derived events through the output adapters.

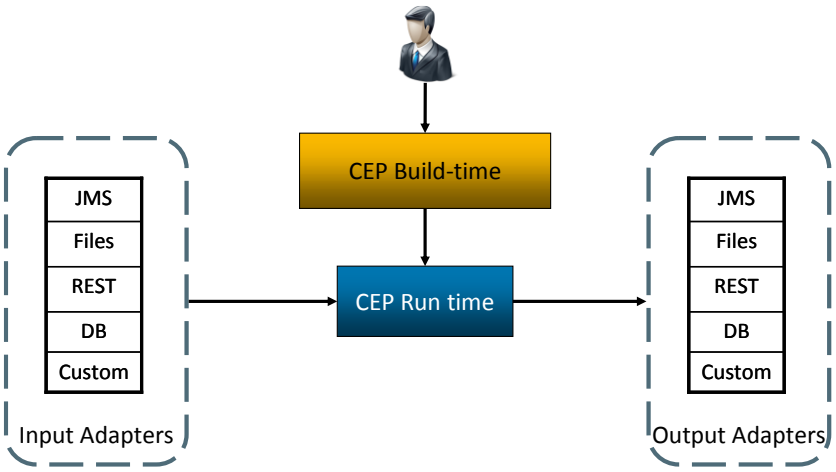


Figure 1: EPM interfaces

2.2.1. Input and output adapters

The definitions of the producers and consumers are specified during the application build-time and are translated into input and output adapters during execution time. The physical entities representing the logical entities of producers and consumers in the CEP are adapter instances. For each producer an input adapter is defined, which defines how to pull the data from the source resource and how to format the data into the CEP's object format before delivering it to the run-time engine. The adapter is environment-agnostic, but uses the environment-specific connector object, injected into the adapter during its creation, to connect to the CEP runtime.

The consumers and their respective output adapters operate in a push mode – each time an event is published by the runtime it is pushed through environment-specific server connectors to the appropriate consumers, represented by their output adapters, which publish the event in the appropriate format to the designated resource.

In FInest, the sources of events can be backend systems (such as AIS or booking systems) or external sensors (e.g., RFID tags) or events originated in FInest during execution and propagated to the CEP module by the BCM. Consumers in FInest are the BCM module (e.g. notification regarding change in shipment status) or the FInest frontend for proactive notifications. The set of rules applicable to the specific TCP (Transport Chain Plan) along with corresponding parameters are passed to the CEP engine during execution by the BCM, which in turn gets its information from the TPM (Transport Plan Module).

2.2.1.1. EPM RESTful API

As mentioned previously, the CEP has three main interfaces: one for getting input events using input adapters, a second for sending output events using output adapters, and a third for getting application specific definitions. In the case of a RESTful API for the first two interfaces, the

CEP engine can activate RESTful services of external applications for receiving input events and for sending output events using its input and output adapters.

The EPM module will build upon the FI-WARE CEP GE[2]. The run-time engine supports a RESTful, resource-oriented API accessed via HTTP that uses either XML-based, JSON-based, or tag-delimited format representations for getting input events and for sending derived events. The CEP engine uses a RESTful client input adapter that is capable of accessing a RESTful web service and pulling input events using its GET method and a RESTful client output adapter, which is capable of accessing a RESTful web service and pushing output events using its PUT method.

The CEP GE uses a REST input adapter that activates a REST service as a client, allowing the CEP GE REST input adapter to access the REST web service declared by the event producer and pull events using the GET method. The CEP GE uses a REST output adapter that activates a REST service as a client, allowing the CEP GE REST output adapter to access the REST web service declared by the event consumer and push events using the PUT method.

In terms of representation format, the CEP REST adapter supports XML-based, JSON-based or tag-delimited formats for the received input event. The format is specified in the CEP producer definition (for pulling input events from it) and in the CEP consumer definition (for pushing output events to it) and is passed as part of the requests using the Content-Type header.

Operations

Getting Events API

The CEP activates a REST API for getting incoming events in a pull mode. The CEP plays as a REST client in this API:

Verb	URI example	Description
GET	/application-name/producer	Retrieve all available incoming events

/application-name/producer

Retrieve all available incoming events from a producer.

In tag-delimited format:

```
GET /application-name/producer
Accept: text/plain
```

Sample result:

```
Name=ShipPosition;ShipID=RTX33;Long=46;Lat=55;Speed=4.0;Time=1333033200;
Name=ShipPosition;ShipID=JF166;Long=47;Lat=55;Speed=2.0;Time=1333033260;
```

Sending Events API

The CEP activates a REST API for sending output events (in a push mode). The CEP plays as a REST client in this API:

[2] http://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/FIWARE.ArchitectureDescription.Data.CEP

Verb	URI example	Description
PUT	/application-name/consumer	Send an output event to a consumer

/application-name/consumer

Send an output event to a consumer

In tag-delimited format:

```
PUT /application-name/consumer
Content-type:text/plain

Name=ShipStopped;ShipID=RTX33;Long=46;Lat=55;Speed=4.0;Time=1333033200;
```

2.3. Definition of FInest complex event processing application

Currently in FInest, the definitions JSON file is created manually and fed into the CEP run-time engine.

The building blocks of a CEP application are:

- Event type – the events that are expected to be received as input or to be sent as output. An event type definition includes the event name and a list of its attributes.
- Producers – the event sources and the way CEP gets events from those sources.
- Consumers – the event consumers and the way they get derived events from the CEP.
- Temporal contexts – time window contexts in which event processing agents are active.
- Segmentation contexts – semantic contexts that are used to group several events to be used by the EPAs (Event Processing Agents).
- Composite contexts – grouping together several different contexts.
- Event processing agents – patterns of incoming events in specific context that detect situations and generate derived events.

The JSON file that is created at build-time contains all EPN definitions, including definitions for event types, EPAs, PRAs, contexts, producers, and consumers. At execution, the run-time accesses the metadata file, loads and parses all the definitions, creates a thread per each input and output adapter, and starts listening for events incoming from the input adapters (producers) and forwards events to output adapters (consumers).

2.4. FInest EPM instantiation

In FInest a set of rules is defined and provided to the run-time engine during the build-time. At execution time, the instantiation of the relevant EPAs is done by specifying for each Transport Execution Plan (TEP) in a single TCP (Transport Chain Plan) the set of rules from the rules definition that apply to the specific TEPs along with the corresponding parameter values. This

set of rules is transferred to the EPM by the BCM, which in turn gets its information from the TPM. The rules activator component in the EPM is responsible for selection and activation of rules relevant for each TEP based on the information received from BCM.

Upon the completion of each TEP within a TCP, the temporal window (temporal context) associated with this TEP is terminated, which causes termination of all EPA instances associated with this context, i.e. all EPA instances related to this TEP. In FInest, a set of rules for the FISH use case has been specified and it is described in detail in D6.4.

3. FInest EPM technical specification

In this section of the report we present the proposed technical architecture of the FInest event processing module that comprises the capabilities discussed so far. We start with a high level overview of the EPM module, followed by a more detailed architecture. Conceptually, the FInest CEP engine can be implemented based on the FI-WARE CEP GE, except for the proactive capabilities that differentiate the FInest CEP from currently available complex events engines.

3.1. EPM high level architecture

The EPM high level architecture is depicted in Figure 2. The CEP runtime engine is the heart of the module – it receives events from producers, monitors and checks them for predefined patterns, and produces derived events. When the latter are emitted outside the module to consumers, they are called detected situations. The entire set of rules to monitor is part of the engine set up and they are defined in advance.

Producers: According to the CEP architecture, producers can produce multiple event types and a single event type can be produced by multiple sources. In FInest, we have identified three sources for events: IoT (Internet of Things), backend systems, and the BCM.

Consumers: Detected situations are consumed by the BCM, which in turn sends notifications to users via the FInest frontend. Proactive notifications are directly sent to the FInest frontend (consumer) by the EPM since these events don't actually alter the execution of the BCM, but serve as meaningful notifications for a user so they can make better decisions (e.g., trigger replanning).

Events: Raw events are introduced into the module by the producers, while derived events are generated as a result of event processing that takes place inside the EPM. Derived events can either be emitted to consumers (detected situations) or can be input for further processing inside the CEP module by other event processing agents. Detected situations can be sent as notifications to users regarding meaningful situations they have subscribed to.

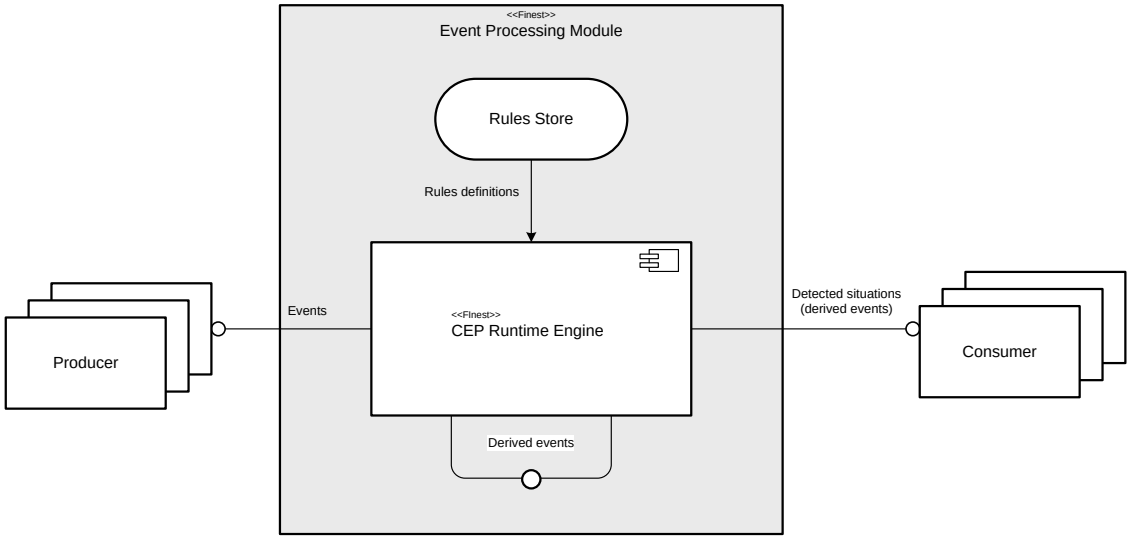


Figure 2: EPM high level architecture

3.2. EPM complex event processing run-time technical specification

Figure 3 shows the internal architecture of the CEP runtime engine.

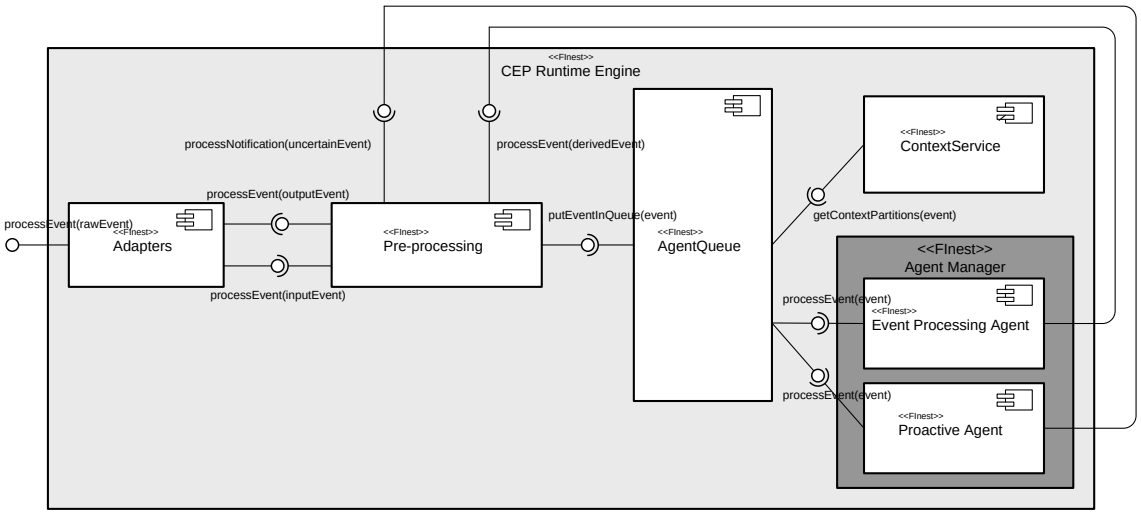


Figure 3: EPM complex event processing run-time architecture

The *adapters component* incorporates both input and output adapter sub-components (see Section 2.2).

- *Input adapters* are responsible for accepting raw events from producers, e.g., from file, JMS (Java Message Service) or REST, converting them into the format that is required by the engine and sending them for preprocessing.

- *Output adapters* receive derived events and proactive notifications and send them to consumers, e.g., to file, JMS, REST.

The *Preprocessing* component is responsible for decisions related to further routing of the accepted event. An event can be routed to a specific agent queue (in case it is an input event) or it can be routed to output adapter (in case it is a derived event) or both. This component preprocesses derived events and proactive notifications and decides whether they will be further processed by the engine (feedback loop) or will be forwarded to output adapters.

The *AgentQueue* component maintains event queues for the different agent types and contexts that these agents are associated with. For each event in a queue, a list of relevant context partitions is retrieved from the *ContextService* (which manages all active context partitions), then the event is sent to event processing/proactive agent instances that are associated with these partitions.

These steps are described in the sequence diagram of Figure 4.

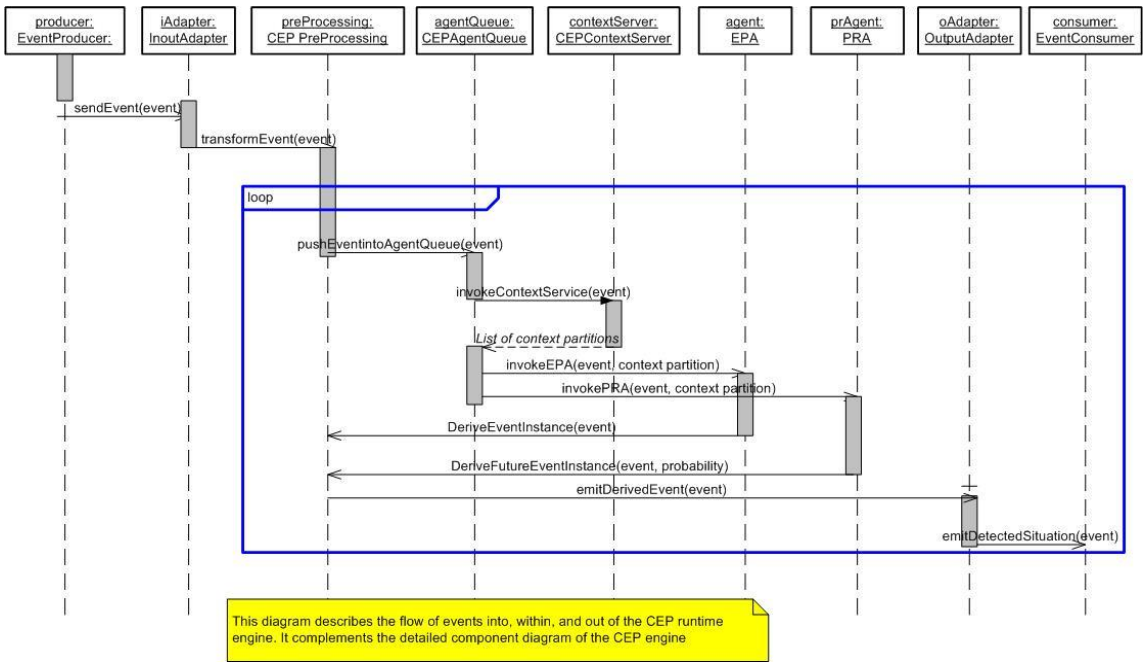


Figure 4: Flow of events into, within, and out of the complex event processing engine

3.3. EPM interfaces with other components

The BCM is responsible for sending the entire set of rules to the EPM. This is performed for each TCP upon shipment initiation. Both rules and events are sent by the BCM to the EPM via a REST API. The EPM activates relevant rules for the entire TCP and each TEP upon TEP initiation event, and deactivates them upon a TEP termination event.

The *Rules activator* is responsible for selection and activation of rules relevant for each TEP based on information received from the BCM.

The CEP engine produces detected situations and proactive notifications to the BCM and the FInest frontend correspondingly.

Figure 5 presents how FInest architectural elements interact with the EPM to realize the event interactions described.

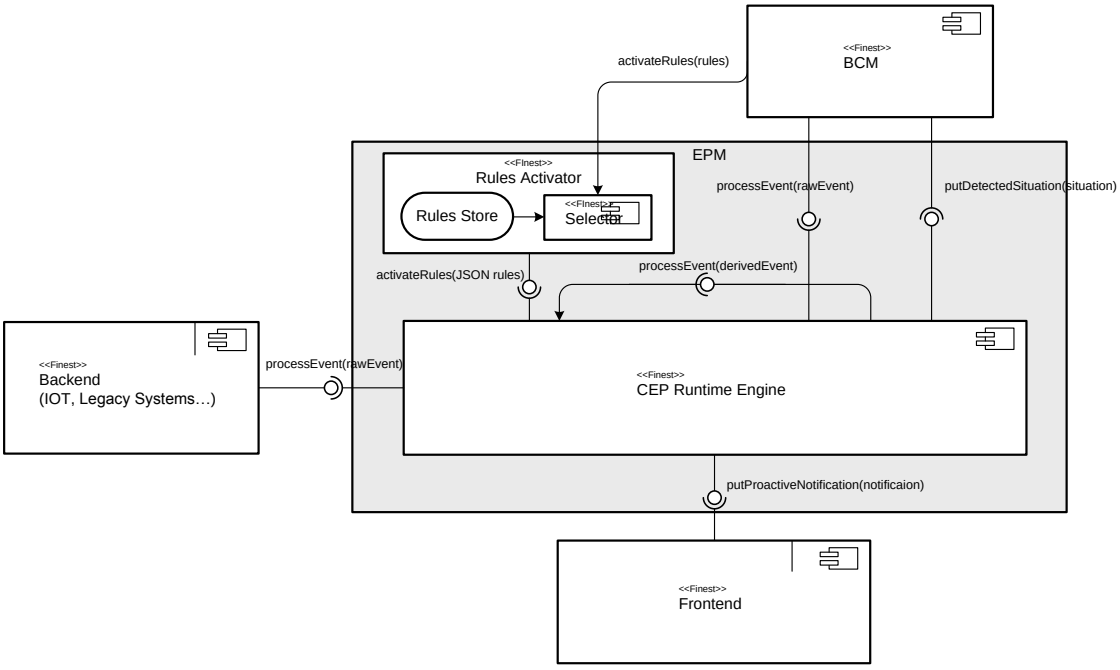


Figure 5: EPM interactions with other components

The steps carried out by the CEP engine upon an arrival of a new set of rules (i.e. initialization of new EPN constructs) is illustrated in Figure 6.

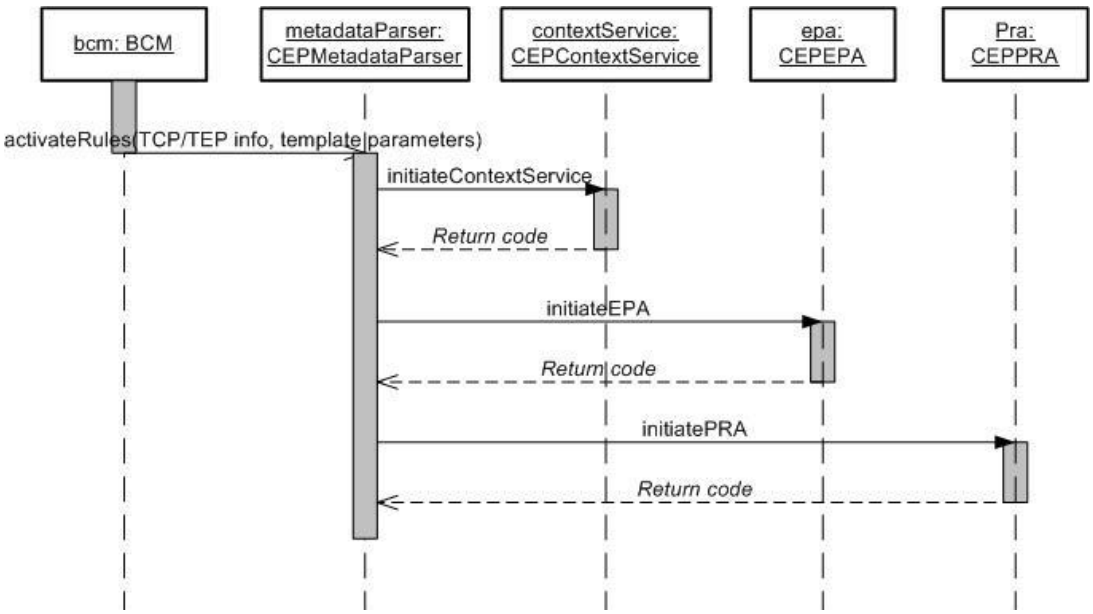


Figure 6: CEP engine set-up

3.4. FInest EPM with FI-WARE GEs

Conceptually, the FInest CEP engine can be implemented based on the FI-WARE CEP GE, except for the proactive capabilities that differentiate the FInest CEP from currently available complex events engine.

Figure 7 depicts the FInest EPM with the implementation of FI-WARE GEs. We note that at this stage we envisage the implementation of two FI-WARE GEs: the CEP engine and the PUB/SUB engine. The Pub/Sub GE will provide the mechanism to get events from external and IoT systems[3].

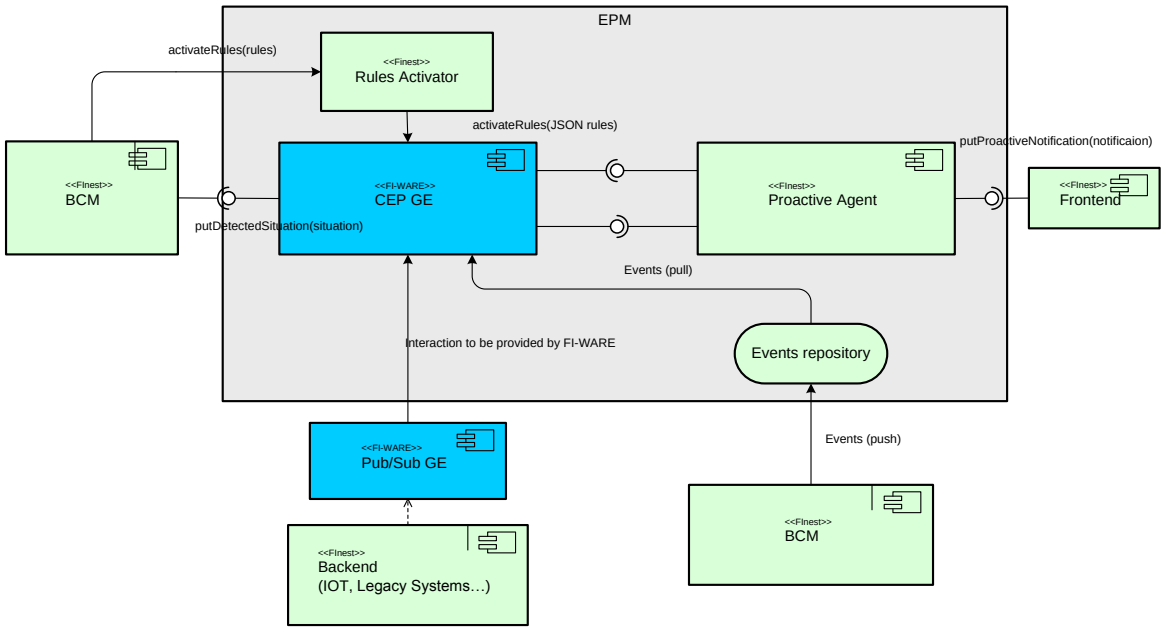
As discussed previously, the CEP GE engine will provide the necessary event processing capabilities except for proactivity. This latter capability will be realized by a specific component that will encompass the specific proactive agents (see Proactive Agent component in the diagram).

The *(FI-WARE) CEP GE* is currently a standalone application (not a web application) that can invoke other components via a REST API for putting produced events; however, other applications can not push events into it in a RESTful manner. The *events repository* component bridges this gap – the BCM pushes events to this repository and they are later pulled by the CEP GE in order to process them.

The *Rules Activator* is also a FInest specific component that is responsible for translating a set of rules from the BCM to a format required by the CEP GE (see also diagram in

Figure 5).

The *Proactive Agent* is a piece of software accountable for processing future events and forecasting detected situations.



[3] http://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/FIWARE.ArchitectureDescription.Data.PubSub

Figure 7: FInest EPM with FI-WARE GEs

3.5. Interfaces data types and definitions

Interactions between various components carry data types as detailed in Figure 8.

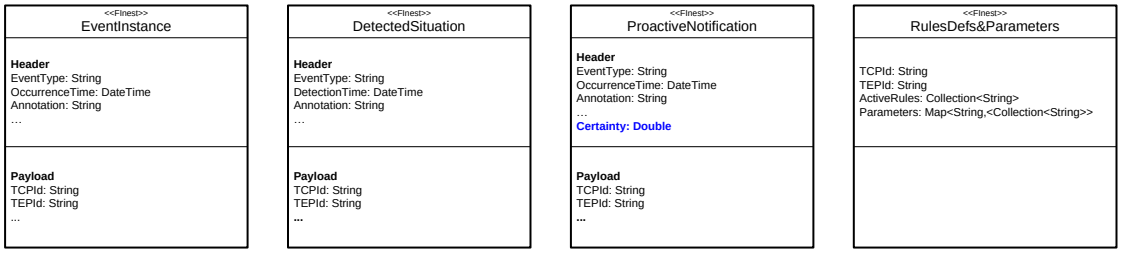


Figure 8: Interfaces and data types

Note that while the EventInstance and DetectedSituation data types have the same structure, the ProactiveNotification also includes the Certainty attribute. ProactiveNotification encloses information regarding an uncertain future event. The *Certainty* attribute is a value in the [0,1] range representing the probability of this event happening (see Section 2.1.1). The ProactiveNotification is an output of the PRA and the Certainty attribute is calculated by predictive models used by this PRA.

In general, the EPM offers three main services or interfaces (the interfaces with its corresponding methods are shown in Figure 9):

- The *EventSourcing interface*, that allows connecting event sources to the EPM;
- The *RulesActivation interface*, that allows activating pre-defined event types and rules for a specific transport instance; and
- The *EventNotification interface*, that allows being notified about identified detected situations.

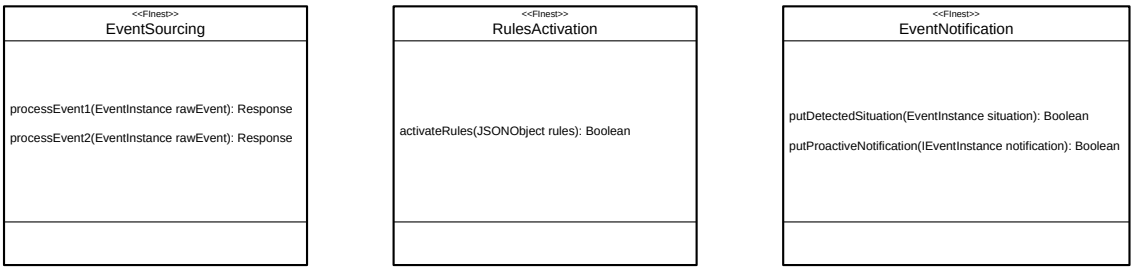


Figure 9: EPM interfaces and methods

The methods/operations grouped by interface are detailed in Table 2 and are shown in the architecture diagrams in Figures 2, 3, 5, and 7. Note that the *source* and *target* columns in the table denote current connections as they appear in the architecture figures. Additional connections (e.g. for phase 2) are possible. For example, proactive notifications could also be connected to other modules in the platform in addition to the frontend.

Table 2: Methods definitions by interfaces

Method name	Source	Target	Description	Data
Interface: EventSourcing				
processEvent1	BCM	EPM	Process raw event (event arriving from BCM), e.g., cancellation of booking, deviation in cargo or schedule, delivery pickup delay etc.	EventInstance - header: EventType, OccurrenceTime, Annotation… - payload: event instance specific attributes
processEvent2	Backend	EPM	Process raw event (event arriving from Backend – IoT and legacy systems), e.g., sensors measurements for container with cargo etc.	EventInstance - header: EventType, OccurrenceTime, Annotation… - payload: event instance specific attributes
Interface: RulesActivation				
activateRules	BCM	EPM	Activate set of rules associated with a specific TCP (and its TEPs). This method is invoked once upon TCP initiation, and it incorporates the entire set of rules to activate + parameters for templates.	RulesDefs&Parameters A list of rules for a specific TCP, including all its TEPs. For templates - parameters values are also provided. This list should be well structured, so that it will be possible to automatically translate rules to (JSON) definition which is an input to CEP engine.
Interface: EventNotification				
putDetectedSituation	EPM	BCM	Put a notification about detected situation to BCM, e.g., a significant deviation in cargo or schedule occurred, a delivery was not picked up as scheduled, import	DetectedSituation - header: EventType, OccurrenceTime, Annotation… - payload:

			license did not arrive for a shipment etc.	Detected situation specific attributes
putProactiveNotification	EPM	Frontend	Put a proactive notification about (uncertain) future event to Frontend, e.g., possible future significant schedule deviation due to high probability to miss a flight.	ProactiveNotification - header: EventType, FututreOccurrenceTime, Certainty, Annotation - payload: Proactive notification specific attributes

4. Phase 2 implementation plan of the event processing module

As previously noted, the EPM will be part of the core B2B collaboration module in the envisioned cSpace platform. More specifically, the FInest EPM has been mapped into the two following Tasks in cSpace:

* Subtask 242: Event processing support in the "Real-time B2B Collaboration" component
* Subtask 451: “Real-time Exception Detection and Handling" baseline application

4.1. cSpace B2B Collaboration module (Task 240)

WP200 "cSpace development" will be accountable for the development of the envisioned cSpace platform components in the form of Tasks as depicted in

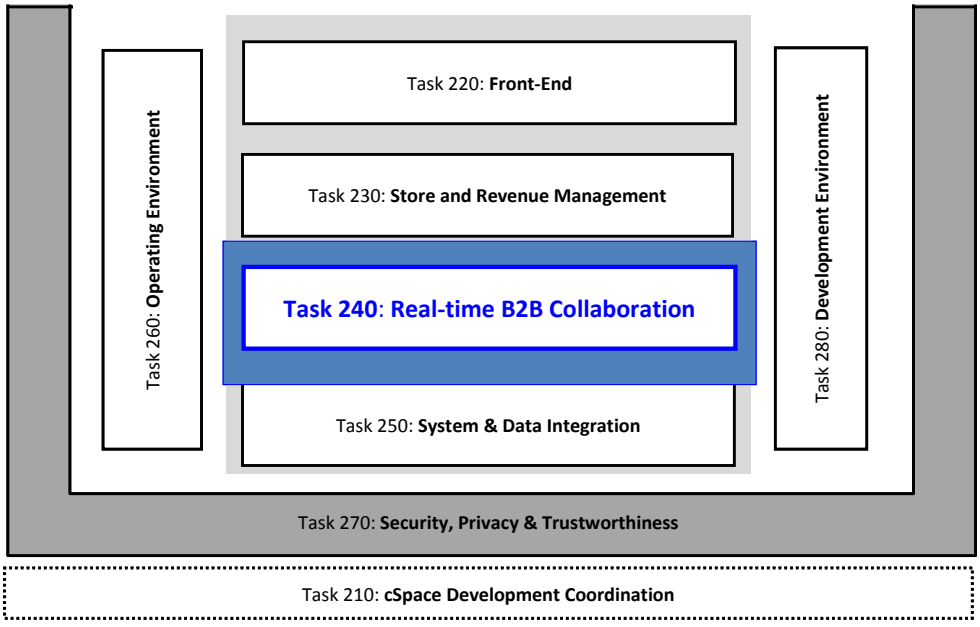


Figure 10. Task 240 "Real-time B2B Collaboration" is, in fact, the natural follow up of the EPM and BCM modules in FInest. This task is concerned with implementing the cSpace real-time B2B collaboration concept, allowing executing, monitoring and controlling collaborative business processes at the heart of cSpace. The aim is to implement the collaboration component based on the artefact-centric approach, and implementing the event processing component based on complex event processing and enhanced it to enable proactive event processing. Task 240 in cSpace includes the development of interoperability components between the collaboration and event processing components to include event-based rule generation, impact of events on collaboration objects, generating appropriate events by the collaboration management component for tracking, and more. In summary, the major outcomes of this task are:

* **Implementation of collaboration management component**, exploiting notion of collaboration objects.
* **Implementation of event management component**, allowing for monitoring and predicting collaborative processes.
* **Integration and implementation of technical interoperability mechanisms** for collaboration management and event management components.

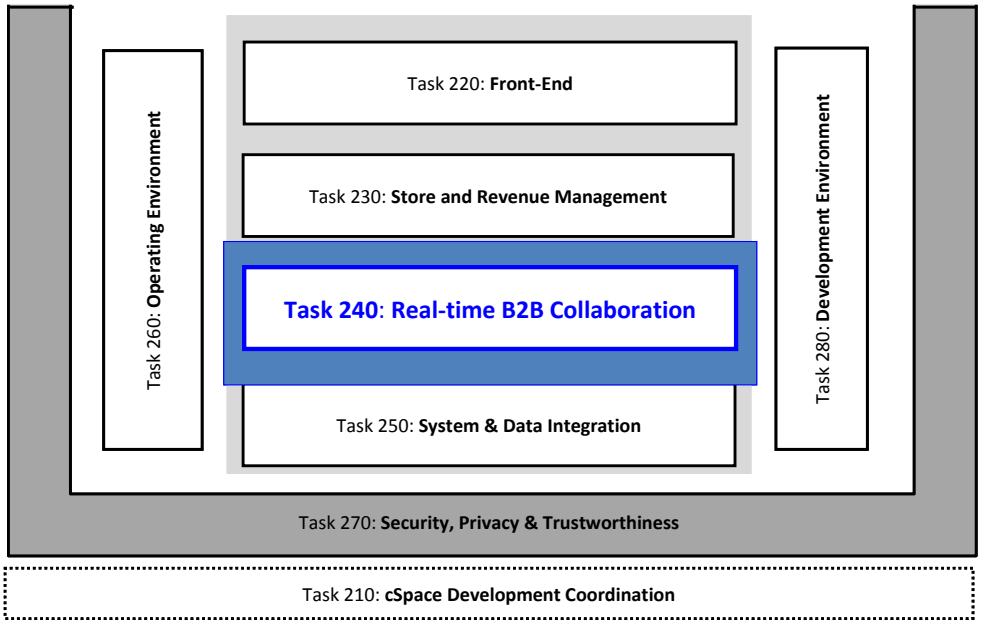


Figure 10: WP200: cSpace development (source: cSpace proposal)

Specifically, the cSpace event processing module will be implemented based on FInest EPM specification as Subtask 242 "Event processing support". This subtask shall include:

* Complex event processing engine based on FI-WARE CEP GE
* Extensions to the CEP engine to enable proactive capabilities
* Configuration and authoring tools to enable the definition of the rules

4.2. Real-time exception detection and handling baseline application

One of the main objectives of cSpace WP400 "Use case trials" is concerned with the development of generic and domain-specific test applications to test and analyse the cSpace offered platform. Task 450 in cSpace targets this objective and is composed of three primary tasks: (i) selecting, developing and deploying cross-domain general “baseline” applications; (ii) aligning and managing application development activities related to the early trials; and (iii) defining technical application requirements for the open call. Baseline applications are applications defined in phase 1, as being of a general nature, which shall add functionalities to the execution of the early trials in phase 2. These apps are based on the services of cSpace, and can be mashed up to other apps to provide enhanced functionalities. Specifically, subtask 451 "Development of the cross-domain baseline applications" is accountable for the implementation of the cSpace baseline applications. Four different baseline applications have been identified so far for implementation: (1) Business services and contract management, (2) Logistics planning, (3) Product information service, and (4) *Real-time exception detection and handling*.

The EPM follow up work will be reflected in the "Real-time exception detection and handling" application by enabling the configuration of a set of rules by defining constraints, observations, and mediations for the application business processes. The Real-time Exception Detection and Handling facility will constantly check the compliance to these constraints and will therewith able to detect potential process violations. Predefined exception handlers will allow an immediate reaction to these deviations, with or without direct user involvement. The definition of constraints and exception handlers can be based on a set of rules, which are defined by the user beforehand. By this, the process monitoring can be adapted to the needs of particular end-users and scenarios. In essence, the Real-time Exception Detection and Handling enables users to define constraints, observations and mitigation actions for business process instances. To this end, the application continuously checks the compliance of these constraints to the actual situation and execution of business processes and thus can – in real-time – detect potential violations. In case of violations, pre-defined exception handlers allow an immediate reaction with short delay. The definition of constraints and exception handlers is supported by a set of rules, which can be defined by the user beforehand. Thereby, process monitoring, tracking and tracing can be adapted to the needs of particular end-users, and even to specific scenarios and tasks.

The cSpace B2B Collaboration Core provides the global knowledge base for all managed business processes and their execution as well as reactive and proactive event monitoring to detect situations of interest. While the cSpace B2B Collaboration Core provides generic mechanisms, more targeted solutions are required that address deviation detection and handling – a requirement identified for most of the trials in cSpace.

Brief summary of main features:

* Adaption of business process monitoring and management to different end-user demands and scenarios, thereby allowing (1) definition observation of user-defined business process constraints, (2) execution of pre-defined exception handlers as immediate reaction to deviations and to mitigate deviations;
* Multi-channel and predictive notification distribution depending on urgency and user demands.

4.3. EPM phase 2 implementation plan Gantt chart

Subtasks 242 and 451 have been outlined on a Gantt chart in Figure 11 along with their respective deliverables. The development of both the event processing module and the base application will be aligned to the project milestones and releases as shown in the Gantt chart. The idea is to have several releases with incremental versions and enhanced capabilities until the final release. We list below the relevant project milestones for each of the two subtasks and also the relevant deliverables along with a brief description.

4.3.1. List of relevant project milestones

Table 3 depicts the relevant project milestones for the two cSpace subtasks related to FInest EPM, i.e. subtasks 242 and 451. For a full description of the milestones refer to D3.4 "Technical specification of domain specific FI platform for transport and logistics and phase 2 implementation plan" (to be submitted at M24) and cSpace proposal.

Table 3: Relevant cSpace project milestones

cSpace Project milestone	Relevant milestones to 242 and 451 subtasks
MS1 Consolidation (M3)	Consolidated conceptual design; detailed release plan; development sup-port facilities set-up Release and development plan for Apps
MS2 Specification (M6)	Public release of cSpace specification (technical design)
MS3 Release V1 (M9)	1st release of cSpace core feature prototypes ready for trials (internal) 1st release of baseline applications and identification of requirements for domain applications
MS4 Trial-Round 1 & Large scale expansion (M12)	Maintenance up-dates of cSpace V1
MS5 Release V2 (M15)	2nd release of cSpace (public) 2nd release of applications
MS6 Trial-Round 2 (M 18)	Maintenance up-dates of cSpace V2
MS7 Release V3 (M21)	3rd and final release of cSpace (public) 3rd release of applications

MS8 Trial-Round 3 (M 24)	Maintenance up-dates of cSpace V3

4.3.2. List of Planned deliverables

For Subtask 242 Event processing support (shown as a blue diamond in Figure 11)

cSpace Technical Architecture and Specification (M6, Type: Report)

The Technical Architecture of the cSpace along with the detailed technical specification of each of its components.

cSpace Integrated Release V1 (M9, Type: Prototype)

The first release (V1) of the cSpace, encompassing the implementations along with usage, guidance, and technical documentation of each cSpace component.

cSpace Development Progress Report and V1 Updates (M12, Type: Report + Prototype)

Report on cSpace development progress with updates on the release plan and on the technical architecture and implementations of the cSpace components from release V1 where necessary.

cSpace Integrated Release V2 (M15, Type: Prototype)

The second release (V2) of the cSpace, encompassing the implementations along with usage guidance and technical documentation of each cSpace component.

cSpace Development Progress Report and V2 Updates (M18, Type: Report + Prototype)

Report on cSpace development progress with updates on the release plan and on the technical architecture and implementations of the cSpace components from release V2 where necessary.

cSpace Integrated Release V3 (M21, Type: Prototype)

The third release (V3) of the cSpace, encompassing the implementations along with usage, guidance, and technical documentation of each cSpace component.

cSpace Development Final Report and V3 Updates (M24, Type: Report + Prototype)

Final report on the cSpace Development activities with updates on the technical architecture and implementations of the cSpace components from release V3 where necessary.

For Subtask 451 Real-time exception detection and handling (shown as a red diamond in Figure 11)

Functionalities of baseline applications (M3, Type: Report)

Full definition and explanation of the functionalities of the baseline applications, documented in the project Wiki.

Baseline applications 1st release (M9, Type: Prototype)

Including the documentation release of the applications in the project Wiki

Baseline applications 2nd release (M15, Type: Prototype)

Including the documentation release of the applications in the project Wiki

Baseline applications 3rd release (M21, Type: Prototype)

Including the documentation release of the applications in the project Wiki

4.3.3. Gantt chart

Figure 11 shows the timeline of the two subtasks related to the EPM in phase 2 of the project along with the relevant project milestones and deliverables.

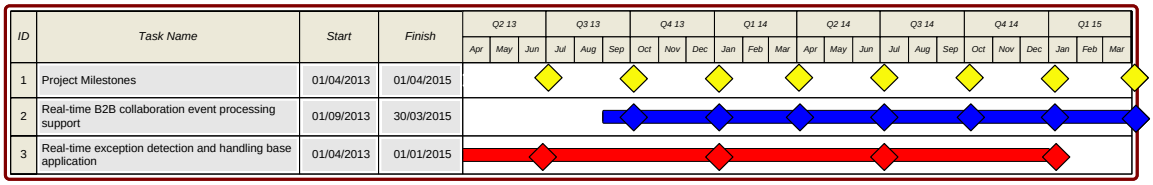


Figure 11: EPM phase 2 implementation plan Gantt chart

5. Summary

This report summarizes the work achieved so far in WP6 "Proactive Event Driven Monitoring" of FInest and reflects its continuation through the cSpace follow up project implementation plan. This report is composed of two main parts, the finalization of the specification of the EPM module in FInest, and the details of its implementation in phase 2 of the project. The EPM module will become one of the two core modules that will constitute the cSpace platform.