D8.3

Stream Data Analysis and Processing Platform First

Document Owner:	Fabiana Fournier (IBM), Inna Skarbovsky (IBM)
Contributors:	Fernando Gigante (AIDIMA)
Dissemination:	Public
Contributing to:	WP8
Date:	9/3/2016
Revision:	3

VERSION HISTORY

NBR	DATE	NOTES AND COMMENTS
0.1	02/09/2015	ToC
0.2	25/11/2015	Revision of input from use cases
0.3	10/12/2015	Input from Fernando Gigante regarding use case
0.4	30/12/2015	First complete draft
0.5	16/1/2016	Added summary and conclusions. First version available for external comments.
1	16/1/2016	Submitted version to internal review
2	1/3/2016	Corrections after internal review
3	9/3/2016	Final version

DELIVERABLE PEER REVIEW SUMMARY

ID	Comments	Addressed (X) Answered (A)

Contents

List of the figures

Acronyms

Acronym	Definition
ASF	Apache Software Foundation
CEP	Complex Event Processing
DSCP	Distributed Stream Computing Platforms
ESP	Event Stream Processing
EPA	Event Processing Agent
EPN	Event Processing Network
JMS	Java Messaging Services
JSON	JavaScript Object Notation
PROTON	IBM PROactive Technology ONline
PSYMBIOSYS	Product – Service sYMBIOtic SYStem

Executive Summary

Work package 8 deals with the selection and construction of tools and platforms for manufacturing intelligence. Within this context, this work relates to tools and platforms suitable for real time analysis of data streams for manufacturing intelligence. The goal is being able to generate, process, and analyze large amounts of streaming data in order to extract meaningful knowledge for a real time and well-founded decisional support.

To this end, we propose a platform to support data stream applications suitable for manufacturing intelligence enriched with complex event processing capabilities. This report presents the architecture of such a platform and a first application based on a scenario related to the monitoring of environmental factors for furniture manufacturing.

Next steps include investigating the applicability of our platform to the requirements of other use cases scenarios in the project.

1 Introduction

1.1 Objectives of the deliverable

D8.3 "Stream Data Analysis and Processing Platform First" objective is to propose an architecture for a distributed streaming computing platform enriched with complex event processing capabilities adequate to support large volumes of streaming data from disparate sources. This report will be followed by a second report to be submitted at month 27 that will extend our demonstrations of the platform to other examples extracted from the use cases in the project.

D8.3 addresses work package 8 "Manufacturing Intelligence Tools and Platforms" objective to develop tools and platforms for the interpretation of real world data streams. While WP8 relates to PSYMBIOSYS toolbox for manufacturing intelligence, this deliverable deals specifically with real time analysis of data streams for manufacturing intelligence. The goal is being able to generate, process, and analyze large amounts of streaming data in order to extract meaningful insights for a real time and well-founded decisional support.

Our proposed framework is based on and leverages the PROTON on Storm open source platform developed by partner IBM in the scope of the FERARI EU project[1]. The proposed framework combines two powerful platforms, a complex event processing tool with a distributive and scalable infrastructure, thus making the proposed framework adequate for scalable distributive environments that require event processing capabilities.

This deliverable is of type *Other*, meaning the main focus is to demonstrate the applicability of our proposed framework for manufacturing intelligence in the context of scenarios taken from the use cases in the project. Our first demonstrator is an event driven application for monitoring environmental factors related to workstations in furniture projects and alerting in case of non-recommended situations at the workstation environment. This report is accompanying by the software and the specific application.

1.2 Relationship with other documents

D8.3 is related to D6.3 "Real World Event Driven Architecture and Events Processing First", that deals with Internet of Things (IoT) technologies suitable for manufacturing technologies, as one of the main technologies is complex event processing. While both reports relate to real-time analysis of data, D6.3 focuses on the core technology and its applicability to IoT, where D8.2 proposes an architecture and a platform to support data streaming as part of PSYMBIOYS toolbox. This deliverable is also related to D2.2 "User Requirements Specifications and Engineering First" as we demonstrate our proposed platform through one of the project use cases. In addition, this deliverable is related to D8.7 "Big Data Analytics Tools and Platforms First". While D8.7 addresses issues of "data in motion" or volume dimension in Big Data, D8.3 addresses "data in motion" or large data streams (of events) and distribution requirements dimensions of Big Data.

1.3 Structure of this deliverable

This report is organized as follows:

- Section 2 provides information on the analysis phase, introduces the role of complex event processing and distributed stream computing platforms;

[1] http://www.ferari-project.eu/

- Section 3 is the core of the deliverable and provides information about the software released including: software description, technical information, and user manual. A specific effort has been devoted in chapter 3.3 to describe a comprehensive configuration and usage of the software in a realistic demonstrator.

2 Analysis phase

In this chapter, the analysis done in the context of the workpackage is reported. The analysis is the starting point for the software selection, understanding, editing, and provision in the context of the PSYMBIOSYS project.

2.1 Complex event processing for real world data streams

In order to understand the role of complex event processing for data streams we first bring a short description of complex event processing and how it developed to cope with requirements of real world data streams.

2.1.1 The role of complex event processing

Gartner defines complex event processing as follows: "Complex-event processing (CEP), sometimes called event stream processing, is a computing technique in which incoming data about what is happening (event data) is processed as it arrives to generate higher-level, more-useful, summary information (complex events). Complex events represent patterns in the data, and may signify threats or opportunities that require a response from the business. One complex event may be the result of calculations performed on a few or on millions of base events (input) from one or more event sources".

Event processing platforms have built-in capabilities for filtering incoming data, storing windows of event data, computing aggregates and detecting patterns. Event processing deals with these functions: get events from sources (event producers), route these events, filter them, normalize or otherwise transform them, aggregate them, detect patterns over multiple events, and transfer them as alerts to a human or as a trigger to an autonomous adaptation system (event consumers). An application or a complete definition set made up of these functions is also known as an Event Processing Network (EPN).

In the PSYMBIOSYS project the complex event processing component (refer to D6.3 - Real World Event Driven Architecture and Events Processing First) is built on and extends the IBM PROactive Technology ONline (PROTON) research asset. This asset has become open source in the scope of the FIWARE FI-PPP project[2] (PROTON being the CEP Generic Enabler in the FI-WARE platform). Open source code as well as technical documentation regarding PROTON can be found in[3]. PROTON comprises an authoring tool, a run-time engine, and producers and consumers adapters. Specifically, it includes an integrated run-time platform to develop, deploy, and maintain event-driven applications using a single programming model. In real-time PROTON executes a JSON (JavaScript Object Notation) definition file that includes all definitions for a specific event-driven application. The JSON file essentially represents the event processing network for a specific application.

According to Gartner [1][2][3][4] and [5], two forms of stream processing software have emerged in the past 15 years. The first were CEP systems that are general purpose development and runtime tools that are used by developers to build custom, event-processing applications without having to re-implement the core algorithms for handling event streams; as they provide the necessary building blocks to build the event driven applications. Modern commercial CEP platform products even include adapters to integrate with event sources, development and testing tools, dashboard and alerting tools, and administration tools. More recently the second form — distributed stream

[2] https://www.fiware.org/
[3] https://github.com/ishkin/Proton/

computing platforms (DSCPs) such as Amazon Web Services Kinesis[4] and open source offerings including Apache Samza[5], Spark[6] and Storm[7] — was developed. DSCPs are general-purpose platforms without full native CEP analytic functions and associated accessories, but they are highly scalable and extensible and usually offer an open programming model, so developers can add the logic to address many kinds of stream processing applications, including some CEP solutions. Therefore, they are not considered "real" complex event processing platforms. Specially, Apache open source projects (Storm, Spark, and recently Samza) have gained a fair amount of attention and interest ([5], [6]) and these may well mature into commercial offerings in future and/or get embedded in existing commercial product sets. DSCPs are designed to cope with Big Data requirements making them an essential component in any organization infrastructure. According to a recent analysis of Forbes[8], the Big Data market will top $84B in 2026.

Today, there are already some implementations that take advantage of the pattern recognition capability of CEP systems along with the scalability capabilities that offer DSCPs, and offer a holistic architecture. The PSYMBIOSYS project is one example of this new approach as explained in the following sections.

2.1.2 The next level – CEP for data streaming

As previously mentioned, as DSCPs lack event processing operators while CEP systems lack inherent support for scalable and distributive applications, there have been several attempts in the last few years to combine between these two. Henceforth, we survey three recent attempts of integrating event processing open source tools with DSCP open source platforms.

2.1.2.1 Streaming-cep-engine

Streaming-cep-engine[9] is a Complex Event Processing platform built on Spark Streaming. It is the result of combining the power of Spark Streaming as a continuous computing framework and Siddhi CEP engine as complex event processing engine (Siddhi is the core engine of WSO2 open source tool). It was first introduced in Spark Summit 2014[10].

WSO2 CEP has also implemented their CEP engine on top of Storm so it can be used in a distributed mode deployment[11]. Storm is an Apache incubation project at The Apache Software Foundation (ASF), sponsored by the Apache Incubator. It is utilized by well-known companies with significant volumes of streaming data, such as The Weather Channel, Spotify, Twitter, and Rocket Fuel.

2.1.2.2 Esper on top of Storm

The storm-esper[12] library provides a bolt that allows using Esper queries on Storm data. Storm's tuples are quite similar to Esper's map event types. The tuple field names map naturally to map keys and the field values to values for these keys. The tuple fields are not typed when they are defined, and considered by Esper as of type Object. In addition, the fact that tuples have to be

[4] http://aws.amazon.com/kinesis/
[5] http://samza.incubator.apache.org/
[6] http://spark.apache.org/streaming/
[7] https://storm.apache.org/
[8] http://www.forbes.com/sites/louiscolumbus/2015/05/25/roundup-of-analytics-big-data-business-intelligence-forecasts-and-market-estimates-2015/
[9] http://stratio.github.io/streaming-cep-engine/
[10] http://spark-summit.org/2014/talk/stratio-streaming-a-new-approach-to-spark-streaming
[11] https://docs.wso2.com/display/CEP400/Creating+a+Storm+Based+Distributed+Execution+Plan
[12] https://github.com/tomdz/storm-espe

defined before a topology is running makes it relatively easy to define the map event type in the setup phase.

The Esper bolt itself is generic. It receives Esper statements and the names of the output fields which will be generated by these Esper statements.

The bolt code itself consists of three pieces. The setup part constructs map event types for each input stream and registers them with Esper. The second part is the transfer of data from Storm to Esper. The execute(Tuple tuple) method is called by Storm whenever a tuple from any of the connected streams is sent to the bolt. The Esper bolt code first has to find the event type name corresponding to the tuple. Then it iterates over the fields in the tuple and puts the values into a map using the field names as the keys. Finally, it passes that map to Esper. At this moment, Esper routes this map (the event) through the statements which in turn might produce new data that it needs to hand back to Storm. For this purpose, the bolt registered itself as a listener for data emitted from any of the statements that were configured in Storm during the setup. Esper then calls back the update method on the bolt if one of the statements generated data. The update method will then basically perform the reverse operation of the execute method and convert the event data to a tuple.

2.1.2.3 PROTON on top of Storm (ProtonOnStorm) in PSYMBIOSIS

IBM partner in the PSYMBIOSYS project has implemented its open source complex event processing research tool PROTON on top of Storm in the scope of the FP7 EU FERARI[13] project, thus making it a distributed and scalable CEP engine. We will capitalize on this effort to include analysis and processing of real time data streams for manufacturing intelligence into PSYMBIOSYS toolbox. Open source code as well as technical documentation regarding PROTON and ProtonOnStorm can be found in[14].

PROTON on Storm architecture is detailed in Section 3.23.2.2 and exemplified with one scenario taken from the AIDIMA use case as presented in Section 3.3.2.

For a background on complex event processing refer to [7] and D6.3.

[13] http://www.ferari-project.eu/

[14] https://github.com/ishkin/Proton/

3 PSYMBIOSYS data streaming platform

This chapter focuses on the description of the software component released. The section starts summarising the overall information about the software released (description, overall data, functionalities, and architecture), after that technical information is reported about architectural stack, technical manual for installation and licensing (including third parties components). Finally the user manual and the conclusions and future steps close the chapter.

Note that this report describes the distributed and scalable version of the data streaming platform. For technical details about the standalone version of the complex event processing component, please refer to D6.3.

3.1 Software description

3.1.1 Overall data

Item	Value
Component Name	ProtonOnStorm
Software version	Version 1.0.0
Reference workpackage	WP8.2
Responsible Partner	IBM
Contact person	Fabiana Fournier (fabiana@il.ibm.com)
Source control	(*)
Short Description	ProtonOnStorm is an integrated platform to support the development, deployment, and maintenance of complex event processing (CEP) applications dealing with large amounts of input events with high rates in a distributed manner.

(*)
https://github.com/ishkin/Proton/tree/master/IBM%20Proactive%20Technology%20Online%20on%20STORM

3.1.2 Purpose of the tool

The purpose of the tool is to enable the design and deployment of event driven applications in a distributed and scalable manner, using a friendly programming model. In the context of PSYMBIOSYS, the CEP PROTON on top of Storm platform enables the development and deployment of event driven applications for some use cases scenarios in the project that deal with high rates and volumes of events. As in the standalone version, the distributed one enables the analysis and processing of input events in real-time and the detection of situations of interest for operational decision making, by using a set of building blocks and a declarative language.

3.1.3 Summary of functionalities

As in the standalone version ProtonOnStorm supports the following event processing features. Note that there is no loss of complex event processing functionalities when using the ProtonOnStorm platform:

- Several types of contexts (and combinations of them): fixed-time context, event-based context, location-based context, and even detected situation-based context. In addition, more than one context may be available and relevant for a specific event-processing agent evaluation at the same time.
- Easy development using web-based user interface, point-and-click editors, and list selections. PROTON uses a declarative language intended for non-programmer users.

- A comprehensive event-processing operator set, including joining operators, absence operators, and aggregation operators.

Technical highlights:

- Is platform-independent, uses Java throughout the system.
- Comes as a J2EE (Java to Enterprise Edition) application or as a J2SE (Java to Standard Edition) application.
- Based on a modular architecture.

The application definitions, i.e. the EPN, are written by the application developer during the build-time. The definitions output in JSON (JavaScript Object Notation) format is provided as configuration to PROTON run-time engine.

The ProtonOnStorm version can process large amounts of input events from distributed sources with high rates.

3.2 Technical information

3.2.1 Storm programming model

Storm is an open source distributed real-time computation system. It provides primitives for processing unbounded streams of data. Key properties of the system are:

- Broad set of use cases: stream processing (processing messages), continuous computation (continuous query on data streams), distributed remote procedure call (parallelizing intensive computational problem)
- Scalability: allowing adding nodes to scale out for each part of topology (the network/application).
- Fault tolerant: automatic reassignment of tasks as needed
- Agnostic programming language: Topologies and processing components can be defined in many languages.

Storm's programming model allows creating custom applications running on top of the platform. The distribution and multi-processing options are provided by the application creator, as part of application authoring and configuration phases, and are handled during runtime by Storm infrastructure.

The relevant Storm primitives include:

- Stream - An unbounded sequence of tuples. Storm provides primitives for processing and transforming streams.
- Tuple – a primitive of the data model. It is a named list of values. A field can be an object of any type, either a primitive or custom defined.
- Spout - a source of streams. For example, a spout can connect to Java Services Messaging (JMS) queue, read messages from the queue, and emit them as stream.
- Bolt – a processing node. It consumes any number of input streams, does some processing, and possibly emits new streams. A bolt can do anything: run functions, filter tuples, do streaming aggregations, streaming joins, talk to databases, and more.
- Topology – the network/application.
    - A topology is a network of spouts and bolts

- Each edge in the network represents a bolt subscribing to the output stream of some other spout or bolt.
- An arbitrarily complex multi-stage stream computation.

To run an application in Storm a topology needs to be created. This is the top level abstraction submitted to Storm cluster for execution. A topology is a computation graph, containing nodes, each of which contains processing logic (spout or bolt), and links between nodes, indicating how data is passed between nodes. Edges in the graph indicate which bolts are subscribing to which streams. When a spout or bolt emits a tuple to a stream, it sends the tuple to every bolt that subscribed to that stream. Figure 1 shows a topology example.

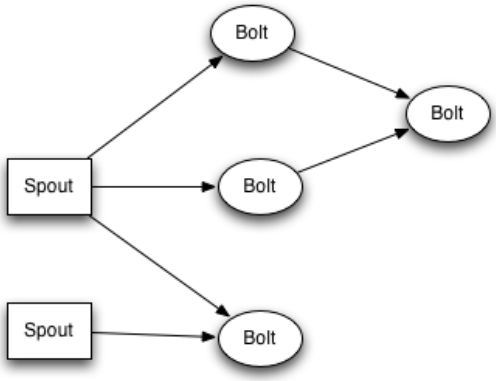

Figure 1: A topology example

- Stream grouping – tells a topology how to send tuples between two components each of which executes as parallel tasks. Figure 2 shows an example of stream grouping.

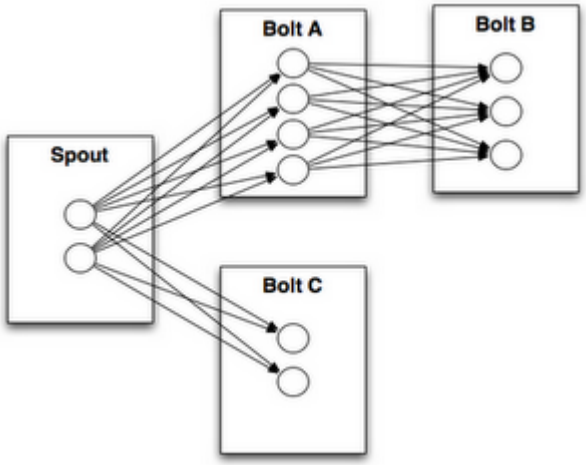

Figure 2: Stream groupings between bolt instances

Storm provides primitives for authoring custom distributed applications. However, as aforementioned, it does not provide complex event processing building blocks. To implement a complex event processing application on top of Storm, concepts such as temporal windows, segmentation contexts, and logical operators have to be coded and implemented for each and every application. To address this gap, we have accommodated PROTON's architecture so it can be implemented on top of Storm. This hybrid/integrated architecture enables the development of scalable and distributed event driven applications without the need to write custom code for any

event-driven logic piece of the application (by using complex event processing building blocks offered by PROTON), or to define and implement distribution functionality (provided by Storm).

3.2.2 ProtonOnStorm internal architecture

PROTON architecture on top of Storm (see Figure 3) comprises the following logical components: the parallelization queues, the context service, and the event processing agent (EPA) manager, which constitute the heart of the event processing system (refer to D6.3). The orchestration of the flow between the components utilizes existing Storm primitives for streaming the events to/from external systems, and for segmenting the event stream.

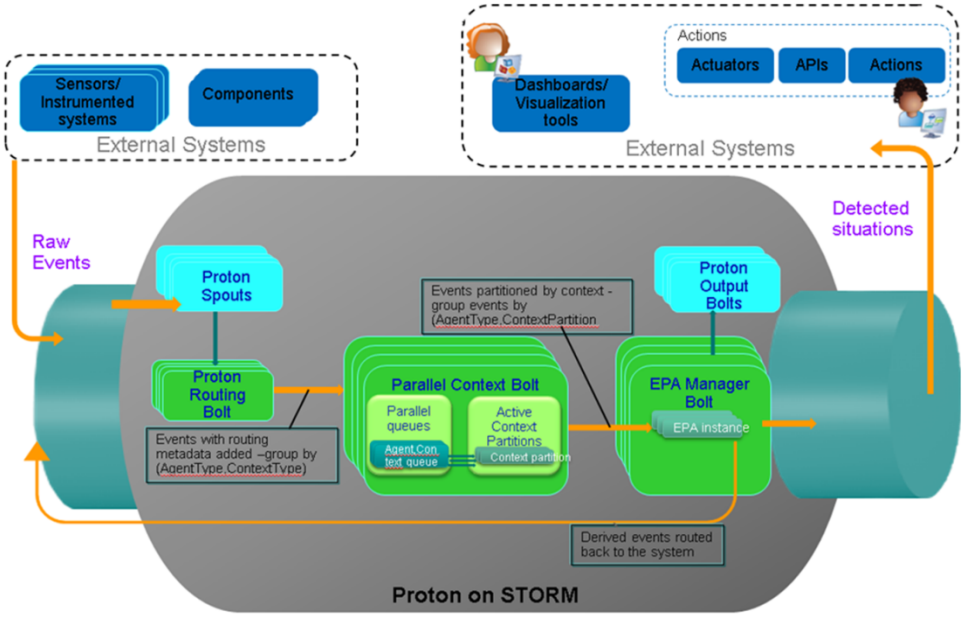

Figure 3- ProtonOnStorm architecture

As part of this approach, the following internal components have been implemented as extensions to PROTON (Figure 4):

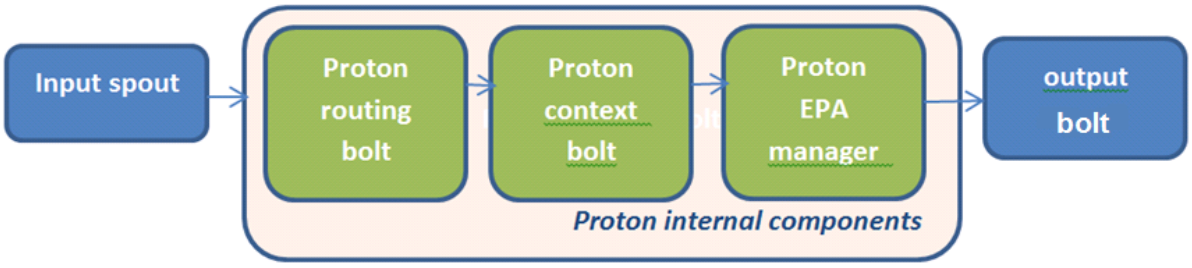

Figure 4: ProtonOnStorm internal components and topology

- PROTON routing bolt: allows to receive incoming raw events/derived events and based on the metadata route them to either consumer logic (via output bolt) or to the PROTON context bolt for internal processing.
- PROTON context bolt: allows to process and to parallelize all events based on their segmentation and temporal contexts.
- EPA manager bolt: allows gathering the state of each EPA instance and applying EPA logical function on this state.

- PROTON topology builder utility: receives an input spout name, an output bolt name, and an initial Storm topology builder, and it creates the CEP part of Storm topology by concatenating the three CEP bolts (see Figure 4) into the given topology.

We use the grouping capabilities provided by Storm to utilize its distribution and scalability functionality (see Figure 5). The fieldGrouping option for grouping event streams is used to firstly route the events based on their metadata to appropriate context instances, and then grouping events streams belonging to the same EPA instance together, and sending them to appropriate instances of EPA bolts.

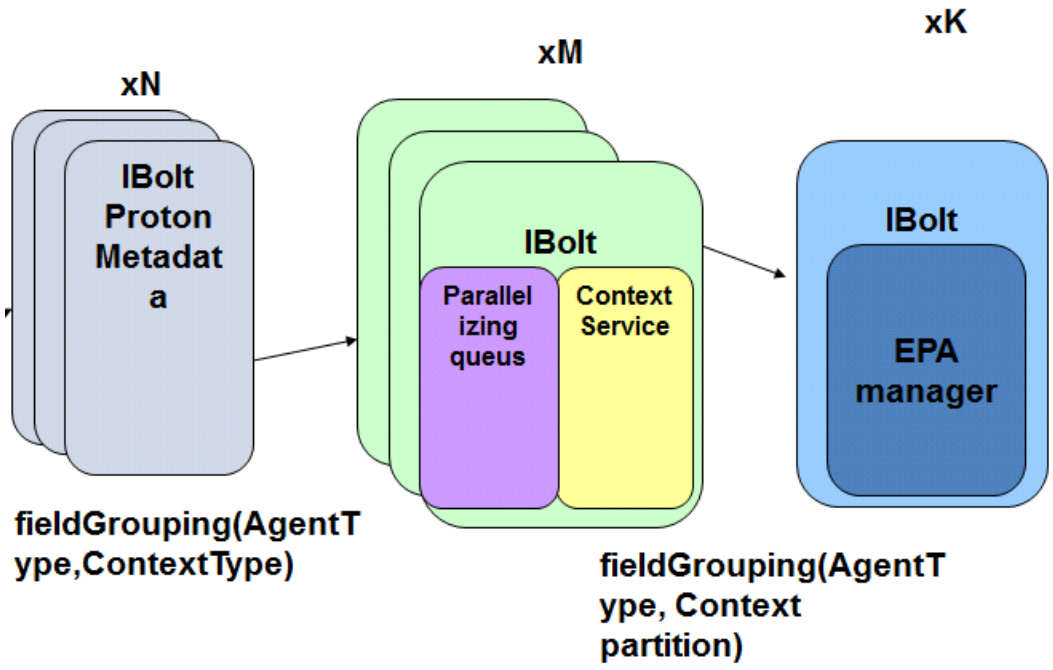

Figure 5: Storm grouping in Proton logical components

The general flow of event processing bolt interactions on Storm is as follows:

1. After the routing metadata of an incoming event is determined by the routing bolt (which has multiple independent parallel instances running), the metadata – the agent name and the context name – is added to the event tuple.
2. We use the Storm field grouping option on the metadata routing fields – the agent name and the context name – to route the information to the next PROTON bolt – the context processing. Therefore, all events which should be processed together – relating to the same context and agent – will be sent to the same instance of the bolt.
3. After queueing the event instance in the relevant queues (in order to solve out of order if needed and parallelize event processing of the same instance where possible by different EPAs in the same EPN) and after processing by context service, the relevant context partition id is added to the tuple.
4. Here again, we use the field grouping on context partition and agent name fields to route the event to specific instances of the relevant EPA, this way performing data segmentation – the event will be routed to the agent instance which manages the state for a specific agent on a specific partition.

5. If the pattern matching is satisfied and we have a derived event, it will be routed back into the system, and passed through the same channels as the raw event (refer to Figure 3).

3.2.3 Technological stack

Item	Value
Nature	Storm application
Programming Language	Java
Development Tools	Eclipse
Additional Libraries	junit,org.testing, org.mockito, org.jmock, org.apache.storm, commons-collections, com.google.guaya, org.slf4j
Application Server	N/A
Databases	N/A

3.2.4 Technical manual

As in the standalone version, the JSON CEP application definitions file can be created in three ways:

1. Build-time user interface – By this, the application developer creates the building blocks of the application definitions. This is done by filling up forms without the need to write any code. The file that is generated is exported in a JSON format to the CEP run-time engine.

2. Programming – The JSON definitions file can alternatively be generated programmatically by an external application and fed into the CEP run-time engine.

3. Manually – The JSON file is created manually and fed into the CEP run-time engine.

The created JSON file comprises the following definitions:

- Event types – the events that are expected to be received as input or to be generated as derived events. An event type definition includes the event name and a list of its attributes.

- Producers – the event sources and the way PROTON gets events from those sources.

- Consumers – the event consumers and the way they get derived events from PROTON.

- 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.

- Composite contexts – grouping together several different contexts.

- Event processing agents – patterns of incoming events in specific context that detect situations and generate derived events. An EPA includes most of the following general characteristics:

    - Unique name

- EPA type (operator). For each operator, different sets of properties and operands are applicable.
- Context
- Other properties such as condition
- Participating events
- Segmentation contexts
- Derived events

The JSON file that is created at build-time contains all EPN definitions, including definitions for event types, EPAs, contexts, producers, and consumers.

In order to implement the application on Storm, several steps to create the additional artifacts need to be taken as described in Section 3.3.2 for our first use case.

Instructions for installation, configuration, and adminstration of ProtonOnStorm can be found at:

https://github.com/ishkin/Proton/blob/master/IBM%20Proactive%20Technology%20Online%20on%20STORM/Proton%20and%20Proton%20on%20top%20of%20STORM.docx

3.2.5 Licensing

Apache 2

3.3 User Manual and use-case

PROTON's user manual can be found at: http://proactive-technology-online.readthedocs.org/en/latest/ProtonUserGuide_FI_WAREv4_4_1/index.html

Specifically, the configuration file for the AIDIMA event-driven application (see following sections) can be found at: http://demos.txt.it:8096/intranet/wp8/m12-deliverables/aidima-usecase-application-defs-and-execution-instructions

3.3.1 Description of the scenario use case

We demonstrate the potential benefits of applying data streaming technologies for manufacturing intelligence in one scenario elaborated with AIDIMA partner in the project. The idea is to test other scenarios and use cases in the project for their applicability to scalable and distributive CEP. In essence the criterion for using the Storm infrastructure and not the standalone version will stem from the requirement of scalability and distribution of the event driven application. Specifically we don't believe at this stage that the scenario exemplified in this work justifies the Storm version, however we deliberately chose one scenario for implementation in both versions of the CEP tool (standalone and on top of Storm) for learning purposes.

Being the application the same, the resulting design, i.e., the business logic, is the same for both PROTON and ProtonOnStorm versions. The implementation, though, is different as described in the next section. For the full EPN, refer to D6.3.

In essence, the idea in the chosen scenario is to provide a comprehensive workplace monitoring and renovation service, as a bundle with the manufactured furniture. The functionalities of the service will allow the definition of optimised and fully customer-oriented renovation project proposals to the customer. The scenario is based on a furniture renovation project of working environments. In this scenario we aim to measure both physiological and emotional factors at the office, in order to design an improve project proposal for the customer. Focusing on the physiological side, we are interested in a monitoring scenario in which factors about the use of the furniture by the worker are

analysed. The system collects information from sensors regarding workstation conditions; however this information is not leveraged to produce real-time alerts about non-recommended situations at the office environment. The aim is to detect potential situations that trigger real time alerts due to physiological factors. To this end, a first EPN has been created with the collaboration of the use case owner with the goal of having something meaningful and representative, yet doable to be achieved in the first year of the project. The outcome is an EPN consisting of five EPAs as detailed in D6.3. For completeness, we bring the three main situations to be detected by the system:

- Temperature at the workplace might cause discomfort as it is too high (*HighTemperatureAlert*)
- Noise level might cause discomfort as it is to high (*NoiseLevelAlert*)
- A worker is sitting at the same workplace for too long (*UserSittingMoreThanOneHourAlert*)

3.3.2 Implementation

To implement the AIDIMA use case on ProtonOnStorm, the following artefacts have been created in Storm (see Figure 6):

- A spout to read events from the CSV file and inject them into the system. In the following versions we plan to connect this spout to the RESTful service provided by AIDIMA for pulling the events and injecting them into the topology
- Storm topology including the input spout, output bolt, and the internal CEP topology
- Output bolt currently writing events to a file, in the future can be upgraded to POST events to a RESTful service or just forward the events to another part of Storm topology. This will be determined based on the use case needs.

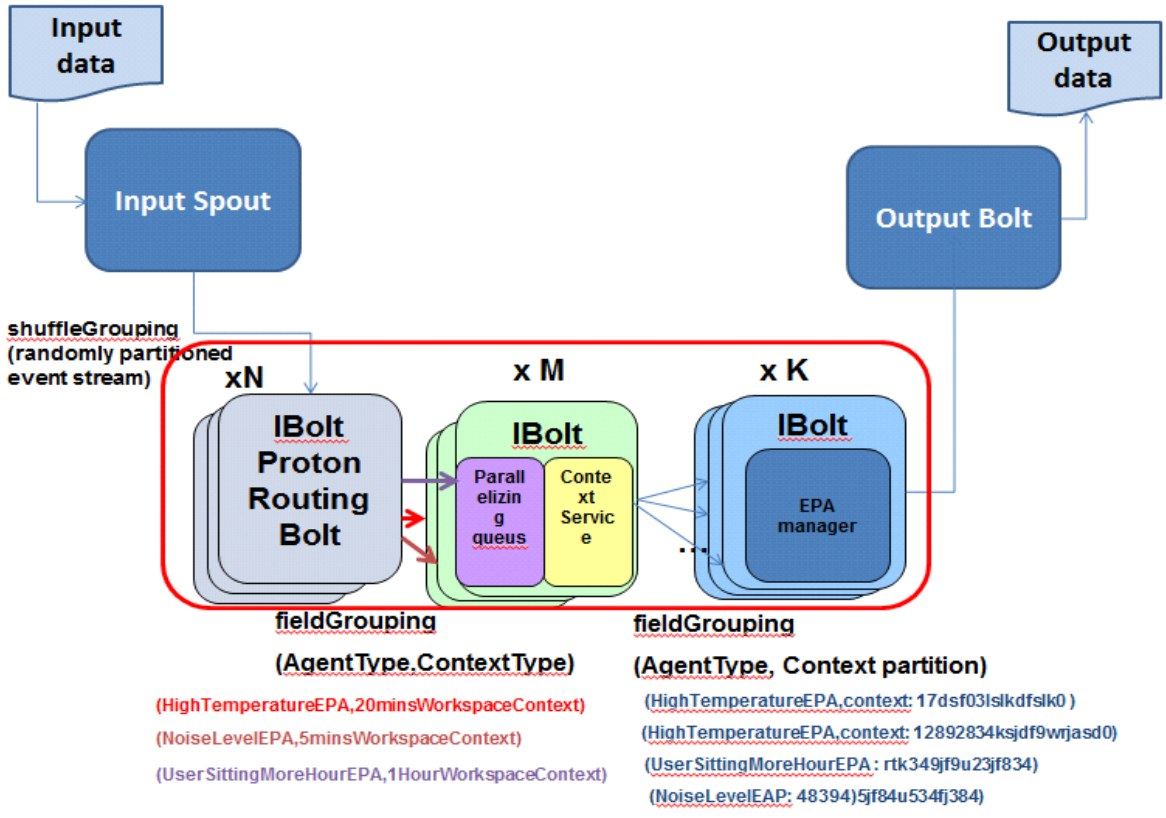

Figure 6: AIDIMA usecase implementation on Storm

Note that although the current input rate and volume do not justify a scalable and distributive platform, we have chosen for a first application something representative and doable to test the applicability of the proposed platform for manufacturing intelligence. The goal is that in the future, the sensors will be connected and the streaming data will be analyzed directly in real-time to give actionable insights, thus requiring this type of platform.

3.4 Conclusions and future plans

In this report we present an architecture for a distributive stream computing platform enriched with complex event processing capabilities. Our proposal is based on and extends the open source ProtonOnStorm platform developed and implemented by partner IBM.

Our proposed platform has been applied with the collaboration of AIDIMA partner in the context of a monitoring scenario in which factors such as noise, pressure, and temperature given by sensors at a workstation are analysed in order to design an improved environment. The scenario has been tested using real data to alert the worker or user of the workstation about potential situations that require their attention, such as, environment which is too hot or too noisy.

Next steps include investigating the applicability of our platform to the requirements of other use cases in the project.

4 References

[1]. Linden A. 2104. Hype Cycle for Advanced Analytics and Data Science. Gartner report G00262076. Published: 30 July 2014.

[2]. LeHong H., Fenn J., and Toit R. L-du. 2014. Hype Cycle for Emerging Technologies. Gartner report G00264126. Published: 28 July 2014.

[3]. Steenstrup K. 2014. Hype Cycle for Operational Technology. Gartner report G00263170. Published: 23 July 2014.

[4]. LeHong H. and Velosa A. 2014. Hype Cycle for the Internet of Things. Gartner report G00264127. Published: 21 July 2014.

[5]. Biscotti F., Schulte W.R., Iijima K., and Heudecker N.. 2014. *Market Guide for Event Stream Processing*. Gartner report G00263080. Published: 14 August 2014.

[6]. Vincent P., 2014. *CEP tooling market survey*. December 3, 2014, [Online]. At: http://www.complexevents.com/2014/12/03/cep-tooling-market-survey-2014/

[7]. Etzion O. and Niblett P. 2010. Event Processing in Action. Manning Publications Company.