Target Outcome: b) Sustainable technologies for a Smarter Society

(FP7 614100)

D6.2. Implementation of Sensor and Data Fusion Module

September 22, 2014 – Version 1.0

Published by the Impress Consortium

Dissemination Level: Public

Project co-funded by the European Commission within the 7th Framework Programme and the Conselho Nacional de Desenvolvimento Científico e Tecnológico
Objective ICT-2013.10.2 EU-Brazil research and development Cooperation

Document control page

Document file: D6.2. Implementation of Sensor and Data Fusion Module.docx
Document version: 1.0
Document owner: Ferry Pramudianto (FIT)

Work package: WP6 Software System Engineering and Context Management
Task: Task 6.2 Sensor and Data Fusion Services
Deliverable type: P

Document status: ☒ approved by the document owner for internal review
☒ approved for submission to the EC

Document history:

Version	Author(s)	Date	Summary of changes made
0.1	Ferry Pramudianto	July 25, 2014	ToC defined
0.2	Ferry Pramudianto	Sept 8, 2014	Content defined
1.0	Ferry Pramudianto	Sept 22 , 2014	Document finalized

Internal review history:

Reviewed by	Date	Summary of comments
Enrico Ferrera	Sept 18 , 2014	Accepted with minor comments
Carlos Kamienski	Sept 18 , 2014	Accepted with minor comments

Index:

1. Executive summary

The IMPRESS development platform consists of a set of technologies that help to build general-purpose applications accessing to a plethora of sources, such as information from the physical world, analyzing and fusing relevant data, and performing monitoring and control operations on complex systems. This is achieved through the definition of a number of tools and pre-defined modules that can be managed and combined in order to define a specific logic flow.

The purpose of this deliverable is to investigate Sensor and Data fusion state of the art and approaches in order to provide a theoretical background. Moreover it reports the progress of the Sensor and Data fusion module of the impress platform, which is being implemented as part of the model driven tool (IoTLink). The sensor fusion module provides several implementation of the sensor fusion algorithms. In addition, it integrates complex event processing engine (CEP), which is able to be configured using query language to process stream of events. CEP is able to identify when certain type of patters occur. Moreover, it also able to aggregate and group sensor data through the aggregation functions. IoTLink also allows developers to define their own sensor fusion algorithms using Java language that provides more flexibility for processing complex data such as image data.

In chapter 3 we presents the theoretical background of the sensor fusion. JDL model is a promising architectural model covering the duality of sensing and control. Several inference and aggregation approaches could be used as a checklist when developing sensor fusion modules, however the concrete implementation must be adjusted according to the use case. In Chapter 4 we presents the implementation of the sensor fusion module as part of the IMPReSS model driven tool. Finally, in Chapter 5 we presents the architecture of the data fusion manager that is generated by the model driven tool.

2. Introduction

1.1 Purpose, context and scope of this deliverable

The IMPRESS development platform consists of a set of technologies that help to build general-purpose applications accessing to a plethora of sources, such as information from the physical world, analyzing and fusing relevant data, and performing monitoring and control operations on complex systems. This is achieved through the definition of a number of tools and pre-defined modules that can be managed and combined in order to define a specific logic flow.

The purpose of this deliverable is to investigate Sensor and Data fusion state of the art and approaches in order to provide a theoretical background. Moreover it reports the progress of the Sensor and Data fusion module of the impress platform, which is being implemented as part of the model driven tool (IoTLink). The sensor fusion module provides several implementation of the sensor fusion algorithms. In addition, it integrates complex event processing engine (CEP), which is able to be configured using query language to process stream of events. CEP is able to identify when certain type of patters occur. Moreover, it also able to aggregate and group sensor data through the aggregation functions. IoTLink also allows developers to define their own sensor fusion algorithms using Java language that provides more flexibility for processing complex data such as image data.

1.2 Background

IMPReSS is a EU-Brazil cooperation project aiming at providing a Systems Development Platform (SDP), which enables rapid and cost effective development of mixed criticality complex systems involving Internet of Things and Services (IoTS) and at the same time facilitates the interplay with users and external systems. The IMPReSS development platform will be usable for any system intended to embrace a smarter society. The demonstration and evaluation of the IMPReSS platform will focus on energy efficiency systems addressing the reduction of energy usage and CO2 footprint in public buildings, enhancing the intelligence of monitoring and control systems as well as stimulating user energy awareness.

The IMPReSS Platform consists of a set of technologies that help to build general-purpose applications accessing to a plethora of sources, such as information from the physical world, analyzing and fusing relevant data, and performing monitoring and control operations on complex system. The IMPReSS project aims at solving the complexity of system development platform (SDP) by providing a holistic approach that includes an Integrated Development Environment (IDE), middleware components, and a deployment tool.

3. Theoretical background

The standardized multi-sensor fusion definition given by the Joint Directors of Laboratories (JDL) (White, 1991) is a "multilevel, multifaceted process dealing with the automatic detection, association, correlation, estimation, and combination of data and information from multiple sources." Data fusion model describes various tasks and processes that need to be considered when fusing data and information. There have been various models defined to get a common understanding of sensor fusion.

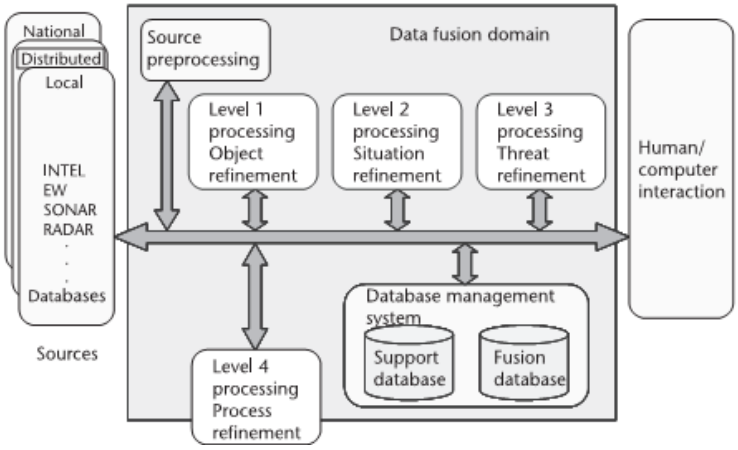

Figure 1. The JDL Model (Liggins, Hall, & Llinas, 2009)

The first generic model was introduced by a data-fusion working group of Joint Directors of Laboratories (JDL), a joint effort within the U.S. department of defense. However, it was intended for decision support system in defense systems. JDL model introduced several components that cover sources, 5 processing levels, a database, and a human computer interaction. The sources receive input from various sensors, a priori knowledge, and human. The database is responsible to maintain the data needed by processes. HCI allows human operators to provide query, input knowledge etc.

The processes consist of 5 levels:

- Level 0 / pre-processing, also known as process alignment or sub-object data association aims at obtaining initial information about target's characteristics by combining the raw data (e.g.: signal, pixel).
- Level 1 processing /object refinement focuses on combining sensor data to estimate position, velocity, and attributes of the observed objects for supporting prediction to future values.
- Level 2 processing / situation refinement tries to describe relationship among the current entities, events and their environment which also includes clustering and relation analysis.
- Level 3 processing / significance estimation tries to project the current situation to the future to draw inference about possible threats, friend or foe vulnerabilities, and opportunities.
- Level 4 processing, / cognitive refinement is a monitoring process of the data fusion performance. This process also tries improving the performance. This is a part of resource management.

Recently, WSN is often used to monitor the behavior of physical objects. To understand the meaning of the e data generated by the WSN, relevant information on how the data is produced must be considered. Context information fusion concerns with how sensor data can be processed

to increase its relevance {Borges, 2013 #500}. As depicted in Figure 2, sensor data that comes as analogue or digital signal is only useful when its relevant information is knows e.g. what does it measure, its location, its precision, etc.

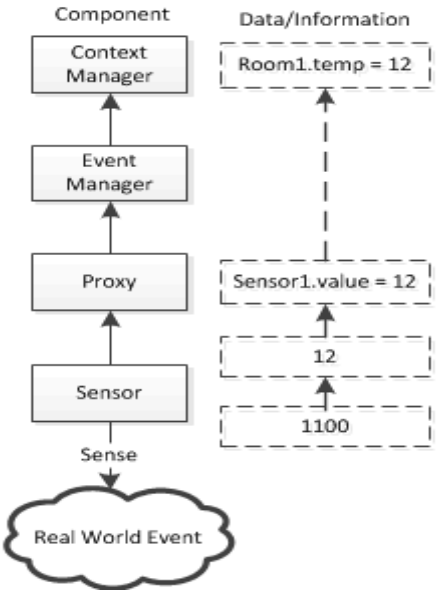

Figure 2. Abstraction of Sensor Data

1.3 Models and Classifications

1.3.1 Information Based

Information based models focuses on abstracting the information handled in the tasks. This model represents the first generation of multi-sensor fusion. The first generic model was introduced by a data fusion working group of Joint Directors of Laboratories (JDL), a joint effort within the U.S. department of defense. However, it was intended for decision support system in defense systems. JDL model introduced several components that cover sources, 5 processing levels, a database, and a human computer interaction. The sources receive input from various sensors, a priori knowledge, and human. The database is responsible to maintain the data needed by processes. HCI allows human operators to provide query, input knowledge etc.

The processes consist of 5 levels:

- Level 0 / *pre-processing*, also known as *process alignment or sub-object data association* aims at obtaining initial information about target's characteristics by combining the raw data (e.g.: signal, pixel).
- Level 1 processing /*object refinement* focuses on combining sensor data to estimate position, velocity, and attributes of the observed objects for supporting prediction to future values.
- Level 2 processing / *situation refinement* tries to describe relationship among the current entities, events and their environment which also includes clustering and relation analysis.
- Level 3 processing / *significance estimation* tries to project the current situation to the future to draw inference about possible threats, friend or foe vulnerabilities, and opportunities.

- Level 4 processing, / *cognitive refinement* is a monitoring process of the data fusion performance. This process also tries improving the performance. This is a part of resource management.

Nakamura summarized information fusion based on several aspects such as relationship among input data, the abstraction level of the manipulated data during fusion process, and the abstraction level of the input and output of a fusion process

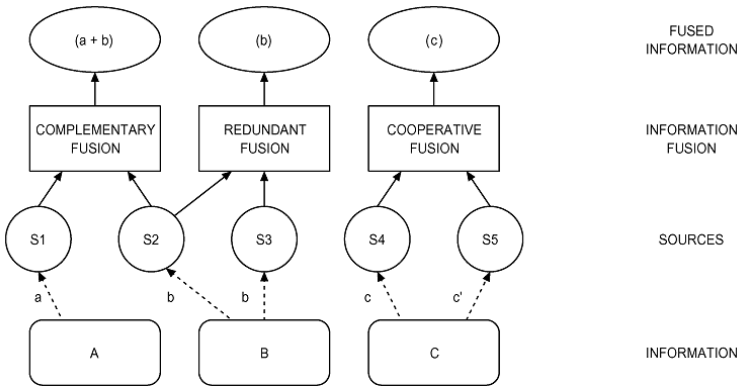

Figure 3. types of information fusion based on the relationship among the sources(Elmenreich, 2002)

Figure 3 depicts, the relation among sources classify fusion to complementary when the information from different sources represents different pieces of a broader scene for instance several temperature sensors which are deployed to cover a wide area, redundant when different sources present the same pieces of information which is used to increase confidence e.g.: sensors, which are deployed with an overlapping range, and cooperative if the pieces of information can derive a new more-complex information e.g.: several type of sensors can infer condition such as storm, which none of the atomic sensor can provide.

Luo proposed four level of information fusion abstraction: signal fusion that deals with n-dimensional input data of sensors, pixel fusion that deals with images, feature fusion which extracts attribute from data, and symbol fusion that take decision of object recognition based on information (Luo, Chih-Chen, & Kuo Lan, 2002). This classification raises ambiguity since one could also treat pixels in image as a 2-dimensional signal. Dasarathy proposed a Data-Feature-Decision model, which consists of 5 categories based on input and output of the fusion (Dasarathy, 1997): Data In–Data Out (DAI-DAO), Data In–Feature Out (DAI-FEO), Feature In–Feature Out (FEI-FEO), Feature In–Decision Out (FEI-DEO), Decision In–Decision Out (DEI-DEO). Its contribution is that the input and output are abstracted clearly. Moreover, system performance is always tuned by decision block by examining the feedback from other blocks. However it is not clear how the processes in other blocks are improved. Iyengar proposed a 3-level abstraction that extends the previous classification: Low level / signal measurement level fusion, Medium level / Attributes or features fusion, High level / Symbol or decision level fusion (Iyengar, Chakrabarty, & QI, 2001). However, based on Dasarathy's work (Dasarathy, 1997), Nakamura added one more level that is able to process data from different level Multi level when data from different level are fused and can results in any level of abstraction. (E. F. Nakamura, Loureiro, & Frery, 2007).

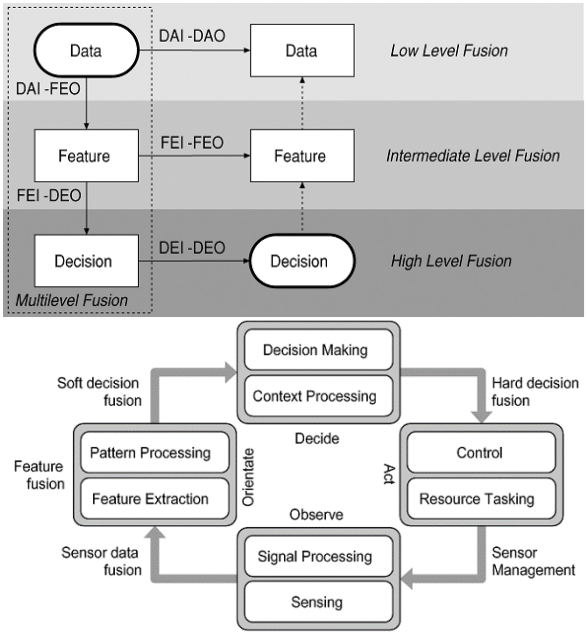

Figure 4. The Dasarathy Model/DFD (left) (E. F. Nakamura et al., 2007) and The Omnibus model (Right) (Bedworth & O'Brien, 2000)

1.3.2 Activity Based

This model concentrates on the abstraction of activities and their sequences of execution. Boyd introduced a control loop cycle, which is known as OODA cycle (observe-orient-decide-act). Information is first gathered in observe phase, then it is fused in the orient phase to get an interpretation of situation, action plan is defined in decide phase, and it is executed in act phase.

Mascolo and Musolesi implemented this model in SCAR algorithm to do routing in wireless sensor network (Mascolo & Musolesi, 2006).

The U.K. Intelligence community introduced a process of developing a raw data into intelligence to support decision making. The activities consist of Collection that gather raw data from environment, Collation that compares, analyses, and correlates the data, Evaluation that fuses data and information, and Dissemination that deliver the result to users who produce decisions.

The disadvantage of the two models is that they only describes the main tasks and does not represent specific tasks of information fusion. Comparing to the JDL model, *Observe* in OODA and *Collection* in Intelligence model correspond to level 0. *Orient* covers level 1, 2, and 3 whereas *Collation* only covers level 1 and *Evaluation* covers level 2 and 3. *Decide* in OODA and *Dissemination* in *Intelligence* match up the level 4. *Act* is not covered by JDL model.

The omnibus model describes the specific tasks of the fusion processes explicitly in a cyclic sequence. The model initially gives an overall perception of the system, which then on the next iterations it goes into more detail subtasks. The omnibus model is enhances of OODA cycle by specifying detail activities representing the fusion tasks that have to be executed in each stage. These activities are similar to Waterfall model.

1.3.3 Role Based

This model shifts the focus on how information fusion can be modeled. In contrast to previous models, role based models do not specify specific task and activities of information fusion explicitly. On the other hand they specify the roles of components and their relationships to

each other. Object oriented model, introduced by Kokar defines four roles: *Actor* who is responsible to interact with real world in terms of collecting information and acting, *Perceiver* assesses information and provides contextualized analysis to the director, *Director* builds an action plan specifying the system goals, and *Manager* controls the actor to execute the plans.

Frankel discussed architecture of human fusion consisting of local and global processes. The local estimator manages the execution of activities based on goals and timetables provided by global process. The global process refines the goals and timetables based on the feedback provided by the local process. Together with Bedworth, he modified the model for machine fusion. The improved model separates control-estimation and goal setting-goal achieving behavior as depicted in Figure 5. Right. The local process act as an estimator trying to achieve goals by going executing the following processes: *sense, perceive, direct, manage, effect*. *Sense* gathers raw data from various sources and sends stimuli to the next process. *Perceive* handles stimuli according their relevance (focus) and informs controller about the stimuli used. *Direct* alerts controller with feedback and receives new goals and timetables (desires). Based on the objectives, *Manage* activates the controller to define what is practical, and then it sends its feedback as expectations about provided decision to the controller. *Effect* executes selected decisions.

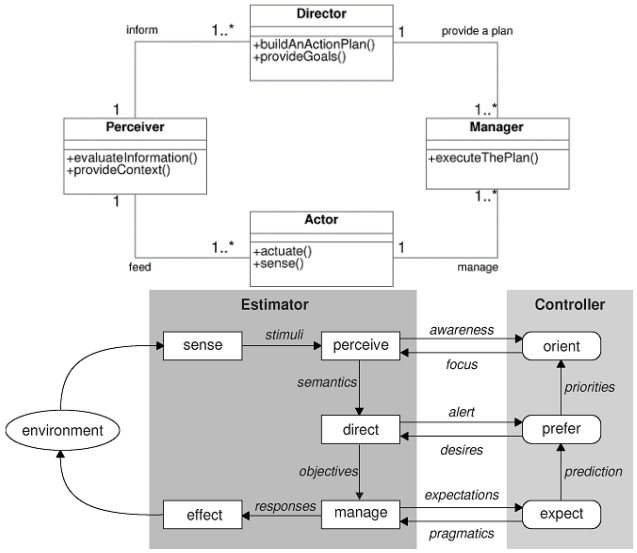

Figure 5. Object oriented model (Left) (Kokar, Bedworth, & Frankel, 2000) and (Frankel & Bedworth, 2000) (Right)

Global process controls the execution of local process. It has three tasks: *Orient* configures the importance of stimuli. *Prefer* prioritizes the most relevant aspects to reach the goals. *Expect* predicts and filters intentional objectives to determine what is practical to the estimator. In practice, the global process will likely be done through human query and definition of operation guidelines representing priorities, desires and pragmatics. Implementation examples of this model are Directed Diffusion (Intanagonwiwat, Govindan, & Estrin, 2000) and TinyDB (Madden, Franklin, Hellerstein, & Hong, 2005).

The role based model contribution is that it defines the actors and their role in information fusion tasks. However this model does not specially address the attributes of service oriented architecture, wireless sensor network, and internet scale distributed system.

1.4 Existing Methods for Information Fusion

Techniques and methods in fuse data involve a various traditional disciplines such as digital signal processing, statistical estimation, control theory, artificial intelligence, and classic numerical methods.

1.4.1 Inference

Nakamura classifies the existing methods based on their purposes into several classes such as inference, estimation, classification, feature maps, abstract sensors, and compression.

The famous inference strategies that are often used in information fusion are Bayesian inference and Demspter-Shafer Belief Accumulation theory. Bayesian inference solves uncertainty by applying conditional probability theory (Bayes Rule) to the evidence obtained from the observations.

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

P(Y|X) represent the posterior probability that wanted to be predicted based on the priors. P(Y) represents the prior probability of Y. P(X|Y), represent the current likelihood of X given Y, and 1/P(X) is a normalizing factor. The problem of Bayesian inference is that P(X) and P(X|Y) have to be estimated or guessed. Bayesian inference has been applied for several works for instance, to reach better accuracy and robustness an advance driving assistance fuses data from different sensors such as laser, radar and video (Coue, Fraichard, Bessiere, & Mazer, 2002), to determine whether an electricity voltage is stable by observing three stability indicators of a power system (Sam, Nwankpa, & Niebur, 2001).

Dampster-Shafer framework generalizes Bayesian theory. The advantage of this framework is, it can be used for incomplete knowledge representation. It is more flexible than Bayesian inference since it allows each source for contributing information with different levels of detail. Moreover, in Dampster-Shafer inference, an a priori probability to unknown preposition is not needed. The probabilities are only assigned when the supporting information is available. However Bracio et al. pointed out that there is a trade-off between the flexibility of Dampster-Shafer and the accuracy of Bayesian inference(Bracio, Horn, & Moller, 1997).

Yu et al. use Dempster-Shafer to build a dynamic picture of battlefield for situation assessment(Yu, Sycara, Giampapa, & Owens, 2004). In DSWare, every decision is associated with a confidence value that is calculated based on Damster-Shafer theory(S. Li, Son, & Stankovic, 2003). Nakamura also used this framework for detecting routing failures and trigger topology construction in his Topology Rebuilding Algorithm(E. Nakamura, Nakamura, Figueiredo, & Loureiro, 2005).

Fuzzy logic uses approximate reasoning to draw conclusions from imprecise premises. Each input is fuzzified by a membership function. The fuzzy rules produce an output that is defuzzified by output rules. (Cui, Hardin, Ragade, & Elmaghraby, 2004) uses fuzzy logic to deal with incomplete and uncertainty of information in a control system for localizing hazardous contamination sources. (Chan Yet & Qidwai, 2005) fused information from ultrasonic sensors to detect potholes and obstacles for an autonomous robotic navigation system. Another example was a work of Gupta et al. and Halgamuge et al. who use fuzzy logic to infer the best cluster heads in WSN. Halgamuge used node concentration, energy level, and centrality as features (Gupta, Riordan, & Srinivas, 2005; Halgamuge, Guru, & Jennings, 2003).

For automatic target tracking (ATR), neural network has also been used since it offers several degree of parallel processing. Baran uses neural network as associative memory guiding the pattern matching process for target recognition (Baran, 2002). Cain et al. classifies targets using neural network based on the information obtained from infrared sensors and a ultraviolet laser radar (Cain, Stewart, & Morse, 1989). Besides, ATR, neural network has also been used for other purposes such as fusing audio–visual information in speech recognition (Lewis & Powers, 2002) and fusing edge maps from image, optical, and infrared sensors(Yiyao, Venkatesh, & Ko, 2001).

Friedlander introduced a In-network inference process by exchanging only the semantic interpretations(Friedlander, 2005). Semantic information fusion needs a knowledge base to infer and fuse data. In his work he explains, that extracting semantic information from sensor network was done by converting sensor data to a formal language. The result is then compared with the known knowledge. He applied this idea to recognize a behavior of a robot based on its trajectory. Whitehouse defined a system that allows users to formulate queries over semantic values. Consequently, the answers are semantic interpretation obtained by the in-network inference processes. Another example is done by Liu and Zhao who decompose declarative queries into graphs that are used for composing services (Liu & Zhao, 2006).

1.4.2 Estimation

Estimation plays an important role in the level 1 of JDL model. Estimation techniques in data fusion reduce the uncertainty of sensed data as a result of sensors physical limitation such as noise, accuracy, reliability, and coverage. the most well known techniques to estimate the raw data includes maximum likelihood, least squares, moving average filter, kalman filter, and particle filter. Maximum likelihood estimate (MLE) tries to find the parameter values that most likely have produced the data. It is suitable to estimate state that is not an outcome of a random variable (Brown, Durrant-Whyte, Leonard, Rao, & Steer, 1992). Xiao et al. improves unreliable communication link in WSN by using distributed and localized MLE, in which every node calculate a local estimate that converge into a global maximum likelihood(Xiao, Boyd, & Lall, 2005). MLE is normally used in location discovery such as obtaining distance, direction, and angle estimations (Lei, Wenliang, & Peng, 2005; Patwari, Hero, Perkins, Correal, & O'Dea, 2003).

Another method based on Bayesian theory is called Maximum a Posteriori (MAP). Similar to MLE it tries to find the most likely value for a state, nonetheless MLE assumes that the state is fixed to unknown parameter space, while MAP assumes the state as an outcome of random variable whose prior probability distributed function is known. Schmitt et al. used MAP for finding the joint positions of a robot and to track positions of moving objects (Schmitt, Hanek, Buck, & Beetz, 2001). Rachlin claims that the implementation of MAP is too expensive for current sensor nodes(Rachlin, Negi, & Khosla, 2006). However, Shah et al. proposed a more efficient implementation by using the estimators as maximum concave function which allows the use of simple numerical maximization algorithm.

Least square is an optimization method that finds the best fits function for a set on input measurement by minimizing the sum of the square error between the values generated by the function and the values obtained from the measurement. Minimizing the square error can be done by using ordinary squared error, Huber loss function, and root mean square error. The least square error is suitable if the parameter to be estimated is fixed. Rabat and Nowak shows that Huber loss function is more suitable in real cases where noisy measurements happen frequently than ordinary square function (Rabbat & Nowak, 2004). Willet used a least squares algorithm in spatial sampling algorithm to define a subset of sensor nodes that provide an initial estimate of the environment(Willett, Martin, & Nowak, 2004). This technique can be used to build a spatial map and mobile node guidance while building the map(Singh, Nowak, & Ramanathan, 2006).

Moving average filter comes from digital signal processing technique. It computes the mean of a measurement values to produce the output signal. The number of measurement to be averaged depends on the window size. Deciding the window size poses a tradeoff between the ability of the algorithm to reduce noise and the ability to detect the change in signal level. Yang et al used this filter for reducing error on target location in a trekking application. Nakamura used this filter to estimate data traffic in WSNs. Another variant of this filter is named Exponentially Weighted Moving Average, which has multiplying factors on each value

and this factors decrease exponentially. Blumenthal used this variant to estimate distances in localization algorithms.

Kalman-Filter is one of the most famous fusion methods. Kalman filter is an optimal recursive data processing algorithm. Kalman-filter finds the most optimum averaging factor for each consequent state. Also somehow remembers a little bit about the past states. Kalman filter incorporates all information that can be provided to it. It uses knowledge of the system and measurement devices, statistical description of the system noises, measurement errors, uncertainty, and any initial condition of the variable of interest. Kalman-Filter is suitable for fusing low level redundant data which has a linear model and its error can be modeled as Gaussian, however when dealing with non-linear models, Extended Kalman Filter or Unscented Kalman Filter should be used. Kalman Filter has been used extensively for source localization and tracking in robotics (Brown et al., 1992) and in WSN it is used to refine location and distance estimates (Hongyang, Deng, Xu, & Li, 2005; Savvides, Park, & Srivastava, 2003) as well as for tracking different sources (T. Li, Ekpenyong, & Yih-Fang, 2006).

Particle filters is also a recursive implementation of signal processing. It is also known as sequential Monte Carlo method (SMC)(Gilks, Richardson, & Spiegelhalter, 1996). Particle filter is an alternative approach of Kalman filter for non Gaussian noise. Particle filter builds a large number of random samples (particle) which is propagated sequentially combining sampling and re-sampling steps. At each time, re-sampling discards some particles to increase the relevance of regions with high posterior probability. In particle filter, each sample is given weight indicating its importance, and the estimate is a weighted sum of all samples of a state. It utilizes 2 phases, prediction and update. in prediction, all particles are modified according to the model. Then in the update, the weight of each particle is re-evaluated based on the actual sensor measurements. Thus, particles with small weight are discarded. Examples of application in sensor fusion include computer vision(Isard & Blake, 1996), multi-target tracking (Crisan & Doucet, 2002), location discovery in WSN (Gustafsson & Gunnarsson, 2003). Aslam used particle filter to explore target tracking using binary detection model (1 bit for moving closer and moving away). Coates used distributed particle filter for target tracking in a hierarchical sensor network(Coates, 2004). Wong investigated a particle filter for fusing multi modality sensors in a target tracking application(Wong, Wu, Ngoh, & Wong, 2004). Sheng proposed two distributed particle filter for multiple target tracking(Sheng, Hu, & Ramanathan, 2005). Another applications in WSN is for determining mobile nodes' locations (Hu & Evans, 2004). Miguez et al. proposed using Monte carlo-particle filter for both target tracking and nodes localization(Miguez & Artes-Rodriguez, 2006).

1.4.3 Feature map

Applications such as guidance systems require spatial information about environment. Feature map provides feature analysis of an environment. An example of feature map includes occupancy grid and network scans. Occupancy grid (also called occupancy map) estimates the occupancy of environment in a multidimensional space. The observed space is divided into multidimensional cells (e.g.: square, cubes) which have probability of being occupied. The probability is computed based on information obtained from sensors, using different inference methods such as Bayesian, fuzzy set, or Dampster-Shafer. Occupancy grid was initially used for ultrasonic sensors determining static environments. Arbuckle introduced a temporal factor in occupancy grid allowing spatial environment being modeled according time properties. Hoover and Olsen model 2D raster as an occupancy map. Another application of occupancy grids include a position estimation, robot's location perception, and navigation. Network scans show the geographical distribution of network resources and activities of a WSN. Zhao used it to retrieve information about residual energy of each node. It work follow: first, it forms an aggregation tree. Second, each node computes its local residual energy and location. When

the energy drop significantly, each node sends the report to the sink. The reports are aggregated based on the region if they have a similar residual energy.

1.4.4 Aggregation

Data aggregation in WSN aims at solving overlap and implosion problems. Implosion happens when data is duplicated many times when it is being routed (e.g.: simple flooding technique). Overlap happens when more sensors disseminate a same data of an event. Aggregation techniques are often used together with data-centric model in WSN such as TinyDB. Intanagonwiwat et al. investigated the latency of greedy aggregation in a dense network. The tradeoff between energy consumption and accuracy was discussed by Boul et al. in aggregation one can suppresses redundant data by discarding duplicates. Another technique for reducing the network overhead caused by frame headers is by placing several reports into a frame such as what TCP window does. This technique is not considered as information fusion, though.

1.4.5 Compression

Compression techniques that utilize information fusion in WSN take advantage of spatial correlation among sensor nodes and the correlation of the sensed data. An example of these techniques is called Distributed Source Coding (DCS) which is extended by Kusuma et al. and Pradhan et al which is called Distributed Source Coding using Syndrome (DISCUS). The main idea is to reduce the number of bits sent from node A to B if their data is correlated. And the possible values of observation can be grouped and indexed. For instance suppose sensor node A and B observation is coded with 3 bit words [000, 001, … ,111]. A and B correlate such that the Hamming distance between A and B is at the most 1. Since the possible states have been grouped and indexed as in Fig.. , A can send only the index (10) to B, and B can infer the value of A based of B's own value and the correlation function (Hamming Distance <=1).

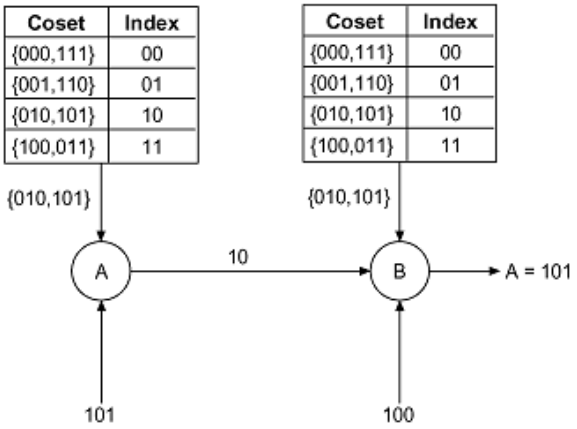

Figure 6. Index table of DISCUS

Hua and Chen improved Viterbi algorithm for DCS that takes advantage of known parity bits at the decoder for error correction. A framework that optimize the tradeoff between compression and network performance for designing and analyzing distributed joint source and network coding was presented by Zhang et al and Ahlswede.

Another compression technique is known as Coding by ordering. The idea is that every node in the region of interest (RoI) sends its data to a border node that collects all data in a big packet that will be sent to the sink. The border node can suppress some packets and sort the rest of the packets in such way that the order indicates the value of the suppressed packets. E.g.: supposed we have 4 nodes (A, B, C, D) whose observation is an integer ranged from 0-5. The border node encodes the packet of node D as follow:

Packet Ordering	Observation from node D
$\{A,B,C\}$	0
$\{A,C,B\}$	1
$\{B,A,C\}$	2
$\{B,C,A\}$	3
$\{C,A,B\}$	4
$\{C,B,A\}$	5

Figure 7. Ordering table

There exist as well other techniques such as EasiPC that reduces redundancy within a single packet to be transmitted, Compressive Wireless Sensing by Bajwa explored a source-channel communication based on compressive sampling (Candes & Wakin, 2008) to estimate sensor data that contain structural integrity. Several approaches use wavelet for compression for instance Ciancio and Ortega decouple data among nodes by exchanging information with neighboring nodes (Ciancio & Ortega, 2004). Tang used wavelet in a source broadcast scheme for extracting significance bits from sensor data (Tang & Raghavendra, 2006).

4. Sensor Fusion Implementation

The "D6.1 Analysis Energy Efficiency Context Sensor Fusion Algorithm" has presented use cases that are relevant for the sensor fusion modules. Scenario 1 describes an energy saving effort by optimizing the elevators in the building depending on the requests. The scenario provides an example to deactivate some elevators during off-peak hours and reactivate all elevators when the requests are high which can be detected from the number of button presses or number of people standing at the elevators' door. Another scenario that was introduced is to control the lighting and air conditioner based on the number of people in the room. In Brazil, split air conditioners are often used instead of centralized cooling system. In large rooms sometimes several split ACs are installed to cool the entire area.

In the university classrooms, the energy consumption can be optimized by considering the occupancy level. When the occupancy is low, the backside of the classrooms could be closed which allow the ACs and the lighting on that side to be switched off to reduce the energy consumption. This enables the system to be able to detect the position of the students in the classroom and suggest them to move to the front side when the occupancy is low.

Considering the use cases in IMPReSS, sensor fusion could be done in different level depending on the intelligent of the devices used in the application. Some sensors only have quite basic function that deliver the obtained signal without putting the data into a proper context. This case requires sensor fusion to be done in different level in order to extract any useful information. On the other hand, most commercial imaging sensors have been shipped with sensor fusion modules, which are able to fulfil certain goals. For instance, Figure 8 shows commercial cameras that are able to count the number of people in the queue by applying image recognition.

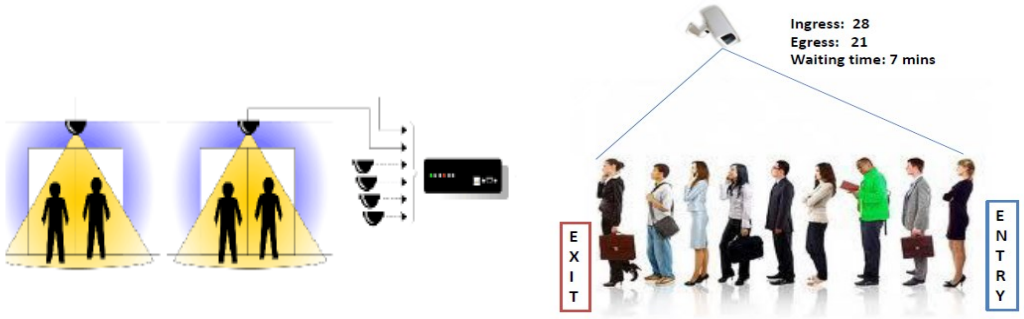

Figure 8. Example of imaging sensors used to count the people in the queue.

These use cases require IMPReSS sensor fusion modules to be able to process data as simple as performing arithmetical functions to numerical values as well as performing complex algorithm such as tracking number of people from imaging data. This means that IMPReSS must provide a predefine sensor fusion algorithms that can ease developers' task processing raw sensor data. In addition, it must allow developers to define their own sensor fusion algorithms when the required algorithms are not available.

1.5 Implementation

IMPReSS implements the sensor and data fusion modules as part of the IMPReSS model driven tool, which is elaborated briefly in D7.2.1. Integrated First Proof of Concept IMPRESS platform. In order to provide the context to the reader, we summarize the model driven tool as follows. The tool is called IoTLink. It enables developers to create a prototype of context aware applications by creating a visual model and generate the software artefacts based on

the visual model. IoTLink allows developers to link the representation of domain objects with sensors and actuators that are able to determine the context of the domain objects. To process the sensor data into a meaningful contextual information, the developers are able to define sensor fusion module that receives the data from the sensors, process them and assign them as the context of the domain objects as depicted in Figure 9.

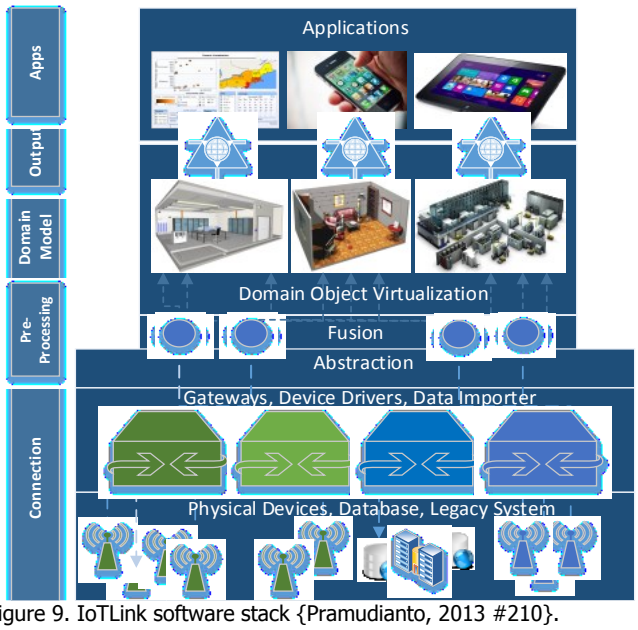

Figure 9. IoTLink software stack {Pramudianto, 2013 #210}.

The first layer at the bottom is responsible for establishing connections to heterogeneous data sources. It represents collection of data sources including physical devices and other subsystems that communicate through gateways or software proxies. This layer is also responsible for providing a uniform interface for accessing the data.
The second layer is responsible for facilitating sensor data processing when required. In this layer, the data can be aggregated and fused using various sensor fusion algorithms to provide a contextual information about the physical objects being observed.
The third layer contains virtual objects that represent the physical objects within the problem domain. Although sensors and actuators are physical objects, in this architecture they are seen as supporting devices that are not a concern of the problem domain. However, sensors are used to determine the properties of the domain objects. Additionally, the actuators are used to carry out the services offered by the domain objects. Thus, linking the sensors and actuators to the properties and functions offered by the virtual objects is required.
In the output layer, the virtual objects then can be exposed to the external applications by communication interfaces or fed to a rule engine, which can be acted upon when the states of the objects have changed. Alternatively, the historical state of the virtual objects can be stored in a database.

The main interface of IoTLink is the main canvas as depicted in the middle of Figure 10. The main canvas contains of five rectangles. The first four rectangles with green and beige colors are containers to group different components that are required for building the context aware prototypes. This separation provide a clear view of the components as well as the flow of the connections among components. The first four rectangles include the following containers (from left to right):

- Connection container
- Sensor Fusion container

- Virtual Object container
- Output container

The last rectangle with a brown color is a placeholder for the template classes. The template classes works similar to classes in an object-oriented programming. The classes define the structure of the objects. This helps developer to maintain changes especially when the number of the object is quite large. By changing the structure of the class, the structure of the objects having the same class are adjusted automatically.

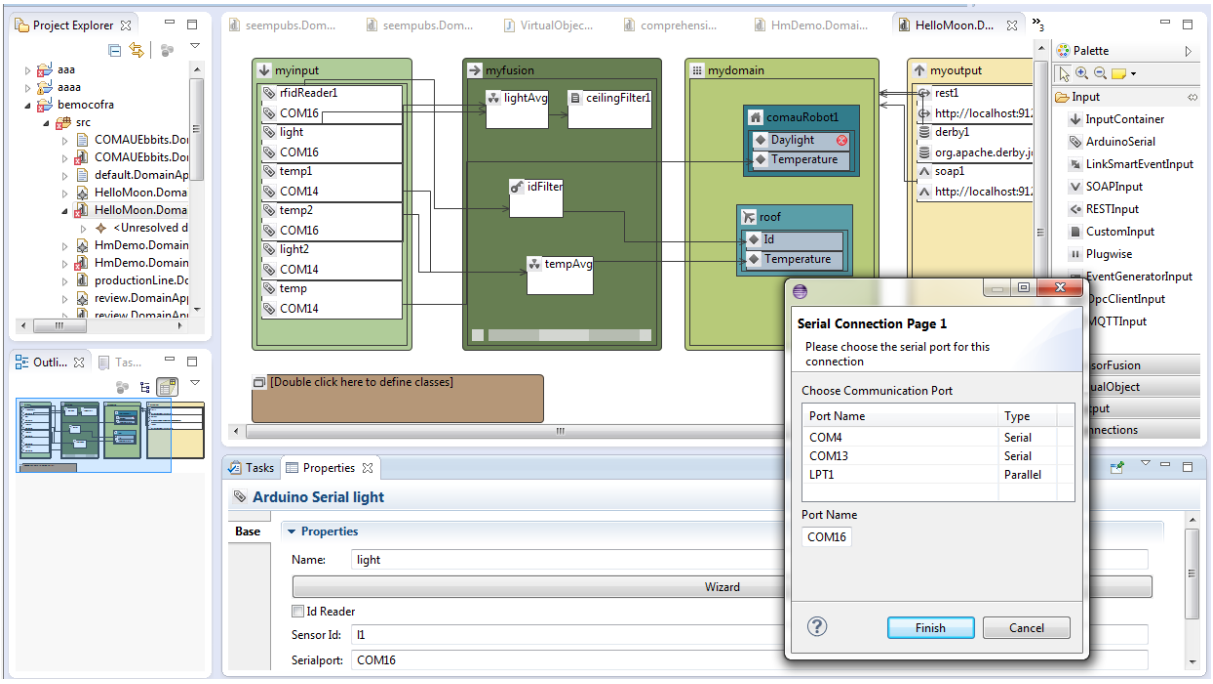

Figure 10. The final iteration of the model development tool

IoTLink provides several predefined sensor fusion algorithms that could be used to process noisy sensor data. When necessary, the sensor fusion modules can be combined as a network of processes as depicted in Figure 9. This allows data to be processed through a network of algorithms until the desired information is extracted. Each of the processes is performed in separate a thread to enable parallel processing of sensor data, which increases the performance of the application particularly on a multicore hardware. Moreover, keeping the processes in separate threads prevent any complex data processing that requires a long time to process, blocking the whole system.
IoTLink is able to build connections to various data sources, which could be physical devices, sensor networks, or software interfaces such as Web services that provide an access to the actual sensors.

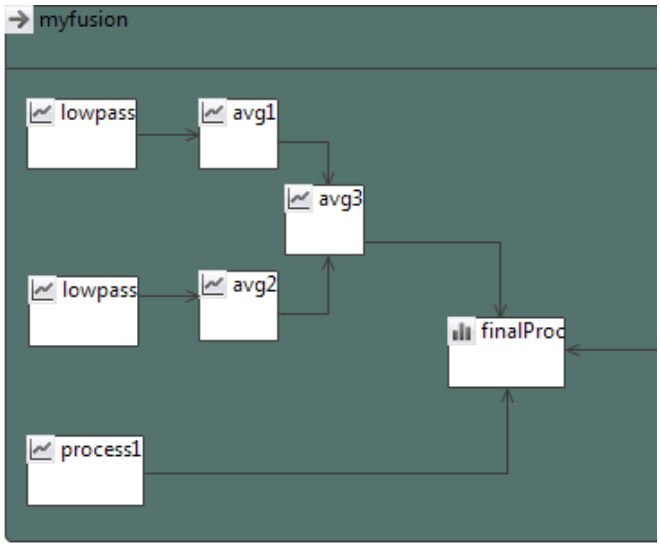

Figure 11. A mesh of sensor fusion modules

The available algorithms that are currently supported by IoTLink are useful for processing simple data type for instance:

- *Moving Average Filter* averages the last N data or the last N data within a period. Moving filter is commonly used for a time series data to observe a long term trend instead of depicting a short term fluctuation which might be caused by sensor noise(Wei, 1994).
- Spatial Average averages the data from a cluster of sensor that have the same type and observe the same events. A cluster of sensor is normally used in critical applications to obtained data with high confidence since providing redundancy could compensate the failure of a sensor by other sensors in the cluster.
- Voting Fusion determines the property of the object based the majority of Boolean value provided by a cluster of sensors observing the same event. Similar to spatial average, the cluster is also used to compensate a sensor failure.
- ID Fusion fuses duplicate ID data that is sent by an ID reader. The algorithm ignores the same id being sent multiple times in a short time frame.

1.5.1 Complex Event Processing

In addition to the implemented algorithms, IoTLink must anticipate that more advance users need a sensor fusion component that can be configured flexibly to extract useful information from the sensor streams. This requirement can be fulfilled by the existing Complex Event Processing (CEP) implementations such as Esper[1], Oracle CEP[2], and Storm[3]. The CEP engines that can be configured to aggregate, fuse, and extract information from streams of events using domain specific language designed specifically for fusing time series data streams. CEP engines are not only used for processing sensor events but also in network security to identify intrusion attempts (Ficco & Romano, 2011), in the banking and financial service to identify e.g. trends in the stock market, credit card fraud (Adi, Botzer, Nechushtai, & Sharon, 2006).

[1] http://esper.codehaus.org/ (Retrieved on August 1, 2014)
[2] http://www.oracle.com/technetwork/middleware/complex-event-processing/overview/complex-event-processing-088095.html (Retrieved on August 1, 2014)
[3] http://storm.incubator.apache.org/ (Retrieved on August 1, 2014)

IoTLink has integrated Esper, which is a leading open source CEP engine whose performance and reliability has been proven by big companies such as PayPal, Accenture, Raython, and Oracle. However, Esper engine is one of many CEP that can be integrated into IoTLink. In the future IoTLink should support more CEP engines that the developers choose.
The complete fusion process is managed by the chosen CEP engines. Therefore the elements needed for a working infrastructure are:

- A component which feeds events into the CEP engines,
- A handler which reacts to the newly created events by the queries,
- An interpreter which holds the queries as processable objects,
- A component which manages these components,
- Finally a standard abstraction of available CEP engines is needed.

After the identification of these roles, we obtain the following architecture (the two CEP engines are presented as examples):

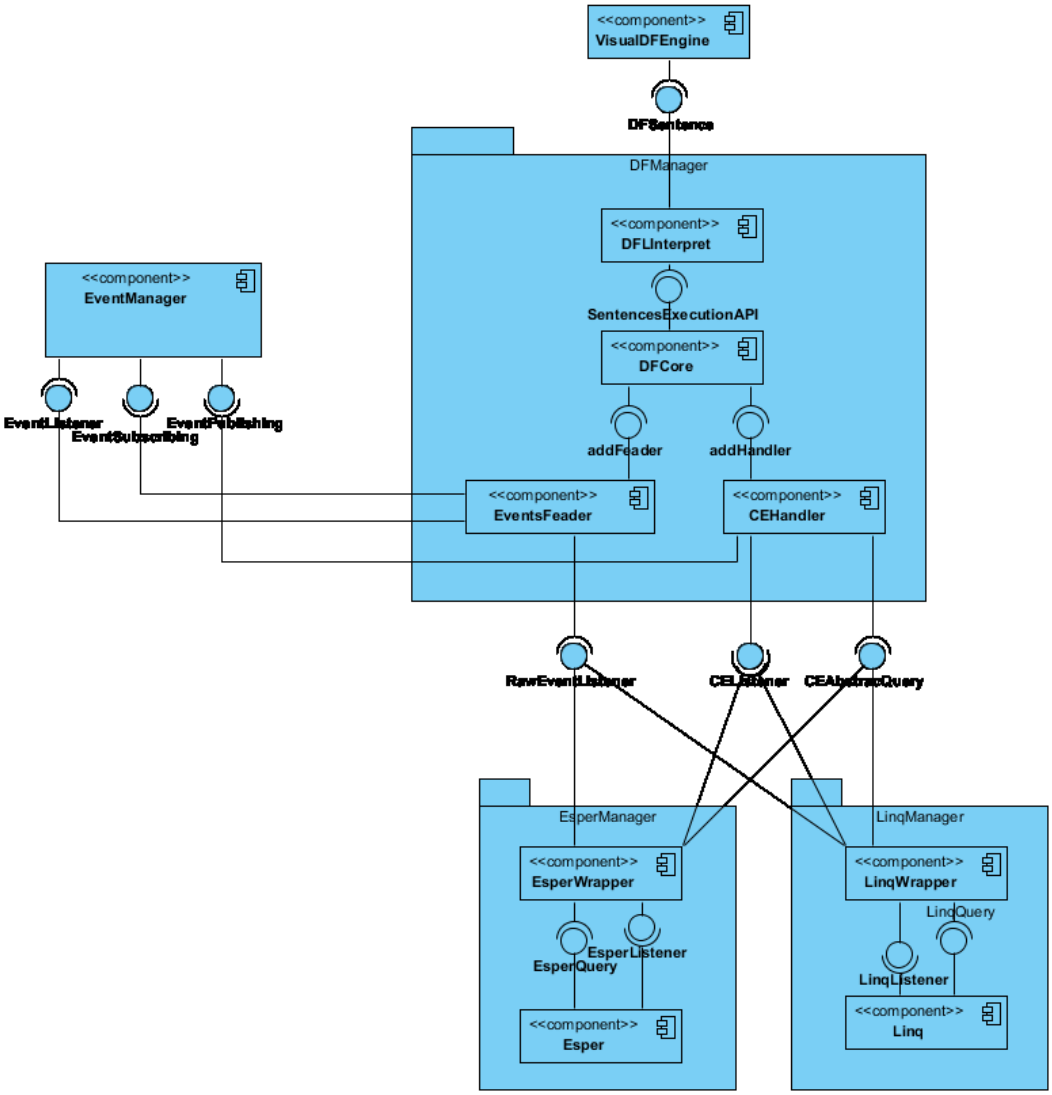

Figure 12: Architecture of first Data Fusion prototype

1.5.1.1 Query Language

Esper takes an advantage of query language similar to SQL, called Event Processing Language (EPL). EPL enables users to query information from the event objects that are inserted to the engine. Since the generated code abstracts sensor streams with the SensorData class, the users are able to query the information from the SensorData objects that have been inserted to the engine. The event streams originating from all data sources are identified by Ids, which can be used to perform processing on that particular stream. For instance Figure 13a shows an example of performing an average on the last five sensor data from a sensor with an Id of "Id1".

```
a) SELECT avg(value) as avgValue FROM SensorData (Id="Id1").win:length(5)
b) SELECT sum(value) FROM SensorData group by timestamp
```

Figure 13. An example of EPL query of a moving window average (a) and sum of the sensor values based on the timestamp (b).

EPL statements determines how the sensor streams should be processed by the engine. EPL Statements may consists of one or more views. Views may represent windows of stream as well as statistics of streams. The example shown in Figure 13a contains a view representing a window of 5 sensor data which is expressed in "win:length(5)".

EPL allows the developers to define the event patterns that the system is interested. In addition, ESPER also allows aggregation and grouping using "group by" and "having" clause, which is useful to e.g. calculate the sum of values based on particular group as depicted by Figure 13b.

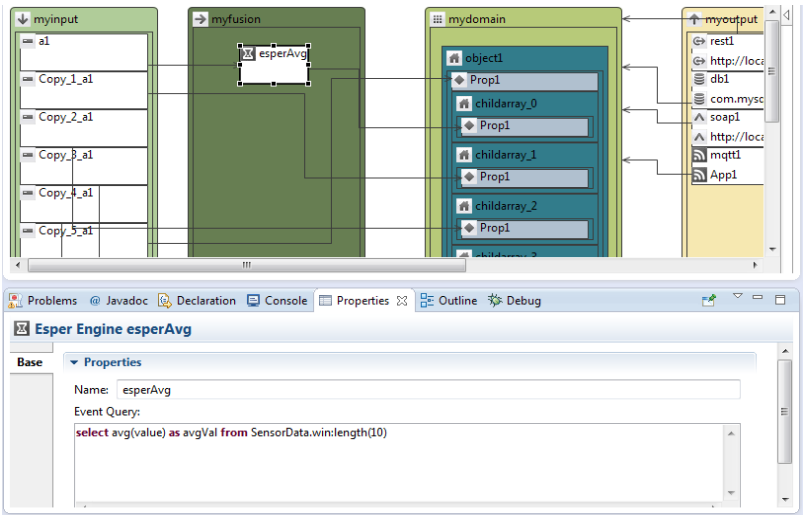

Figure 14. Configuring Esper engine through EPL in the property sheet of the Esper Engine component.

Using Esper engine, the users only need to define EPL statements in the property sheet of EsperEngine as depicted in Figure 14. Similar to other sensor fusion components, each EsperEngine block will be generated as a separate thread, and the result of the thread is pushed to the component that is connected to it e.g. in Figure 14, the last 10 sensor values belong to the "a1" is averaged and pushed to the "Prop1" of the "chlidarray_0" by the "esperAvg" block.

1.5.2 Custom sensor fusion algorithm

Despite of the Esper's flexibility to process sensor stream, it would not be able to cover all possible sensor fusion use cases. For instance, when dealing with picture or sound data, other sensor fusion algorithms must be used. Therefore IoTLink allows the users to define their own custom algorithms in Java language that implements the "SensorFusion" interface. In this case, the sensor data is represented in byte array, which is quite generic to represent digital signals. The users must implement two methods. First, the syncFuse that runs the fusion process in the main thread and blocks the system during the fusion process. Second, which is also the recommended way to use, is to implement the asyncFuse method that must process the sensor data in a separate thread.

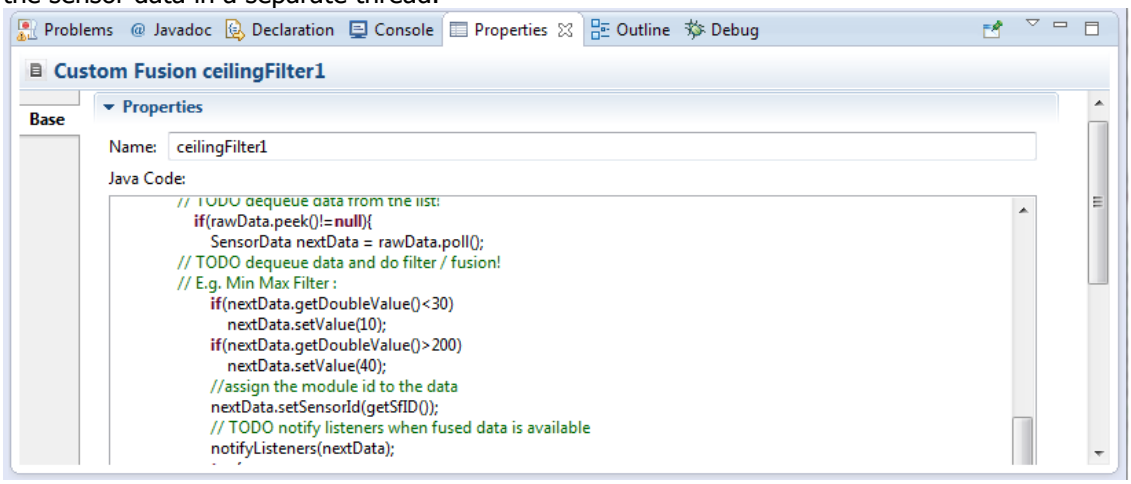

Figure 15. Defining the sensor fusion algorithms in Java.

The code generator generates and initializes the objects of the output components in the MainApp class. The output objects are registered as listeners to property change events of the virtual objects. Acting as property listeners enable them to perform output routines such as writing to database or transmitting the data over a network, when the state of the virtual objects change.

The interaction between the software objects generated by IoTLink is shown in Figure 16. Firstly, the MainApp initializes the concrete objects of the Connection, SensorFusion, VirtualObject, and Outputs. After the virtual objects are initialized, they are stored in a singleton Map within the VirtualObjectFactory classes so that the Input, SensorFusion, and the Output components could retrieve them easily.
The input package consists of connection classes that are responsible to establish connections to the physical devices as well as network protocols such as Web services. Each device type and network protocols may require a specific implementation, which must be implemented as a code template that is used by the code generator to produce the Java code.
When the connection objects are linked to the sensor fusion modules, the incoming data from the connections is pushed to the respective sensor fusion modules. The data could go through several level of fusion depending on how the sensor fusion components are modeled. After the data is processed by the last node of sensor fusion, it is assigned to a property of a virtual object. If the data does not need to be processed through the sensor fusion modules, the connections linked to the properties of the virtual object directly. In this case, as soon as the data from the connection objects are available it is directly pushed to the corresponding domain object and assigned to the property to which connection object is connected.

Next, the generated objects must be chained together according to the links defined in the graphical model. The links are implemented in the MainApp by making the objects as the listeners of the other objects to which they are linked. Chaining these objects through publish subscribe method provides a simple solution and yet quite powerful for ensuring that the data is propagated from one component to another component instantly.

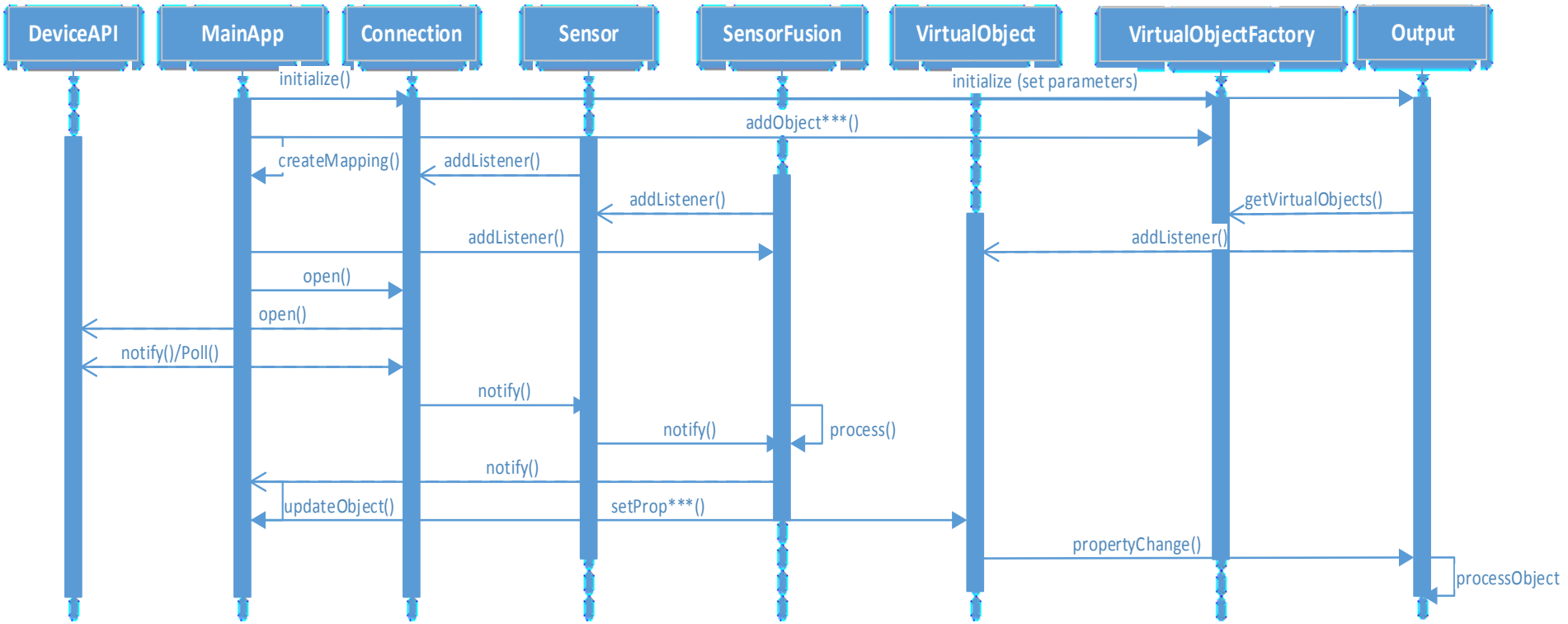

Figure 16. Interaction diagram between the generated classes for updating properties.

5. Data Fusion Manager

The Data fusion architecture is realized by the Data Fusion Manager, implemented as described here.

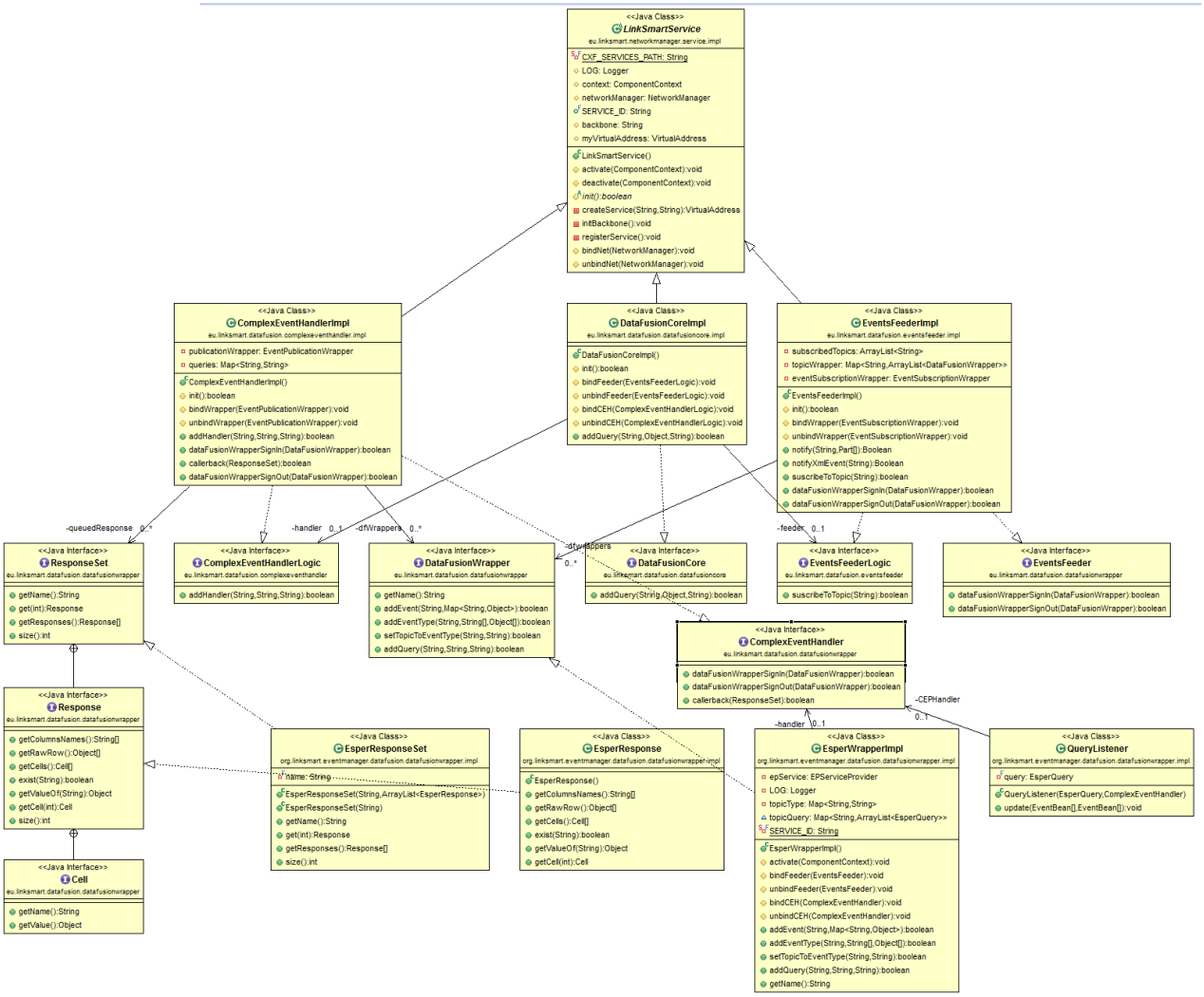

Figure 17: Data Fusion Architecture Implementation

5.1 Data Fusion Core

This component is responsible of the internal logic of the Data Fusion manager. After a query is interpreted and sent by the DFL-Interpreter then DF-Core coordinates and takes the necessary actions for the correct execution of the queries.

5.1.1 Interface DataFusionCore

```
public interface DataFusionCore
```

This interface represents the DataFusionManagement Core. The core is responsible of the internal logic of the DF manager.

- **Method Summary**

Modifier and Type	Method and Description
boolean	**addQuery**(java.lang.String name, java.lang.Object query, java.lang.String topic) Add a query to the manager.

- **Method Detail**
  - boolean addQuery(java.lang.String name,java.lang.Object query, java.lang.String topic)

    Add a query to the manager.

    Parameters:
    - name - of the query. This is how the query will be addressed.

      Also, name indicates where data is going to be published when the query is triggered e.g. "my.event.query" indicates that the event will be published in /my/event/query/

    - query - as Object. The type generated by the interpreter is not yet defined (temporal)
    - topic - where the query will be deployed.

    Returns:
    - true if the query was deployed successfully, false otherwise.

5.2 Events Feeder

This component provides a layer between the event providers and the CEP-engines. This allows a separation of the CEP-Wrappers and the event brokers so one or the other can be exchanged. This means that if the event provider is changed the event feeder should be changed, but not the different wrappers.

Also this component creates a single program logic for the management of several streams and topics regardless of a particular CEP-Engine implementation.

5.2.1 Interface EventsFeeder

```
public interface EventsFeeder
```

This is the part of the API offered by EventFeeder. The EventsFeeder make an abstraction layer between the event provider/s and the CEP-engines.

This is the API offered to the CEP engine wrapper/s.

- **Method Summary**

Modifier and Type	Method and Description
boolean	**dataFusionWrapperSignIn**(DataFusionWrapper dfw) The feeder do not have an awareness of which engines are available.
boolean	**dataFusionWrapperSignOut**(DataFusionWrapper dfw) If a subscriber does not want to be fed events any more then this will have to be indicated explicitly.

- **Method Detail**
  - **dataFusionWrapperSignIn**

```
boolean dataFusionWrapperSignIn(DataFusionWrapper dfw)
```

The feeder does not have an awareness of which engines are available. For a feeder to be able to interact with a Data Fusion Engine, the wrapper of the engine has to explicitly subscribe to the feeder as a Data Fusion engine. Doing so through this function

Parameters:
`dfw` - is the `DataFusionWrapper` to be subscribed to.
Returns:
`true` in a successful subscription, `false` otherwise.

- **dataFusionWrapperSignOut**

```
boolean dataFusionWrapperSignOut(DataFusionWrapper dfw)
```

If a subscriber want to not be fed events any more then this have to be indicated explicitly.

Parameters:
`dfw` - is the `DataFusionWrapper` to be unsubscribed to.
Returns:
`true` in a successful unsubscription, `false` otherwise.

5.2.2 Interface EventsFeederLogic

```
public interface EventsFeederLogic
```

This interface represent the inner logic needed by the core of the API of the feeder.

See Also:
`DataFusionWrapper`

- **Method Summary**

Modifier and Type	Method and Description
boolean	**suscribeToTopic**(java.lang.String topic) Indicates to the Feeder to start feeding the CEP engines with events coming from a topic.

- **Method Detail**
  - **suscribeToTopic**

```
boolean suscribeToTopic(java.lang.String topic)
```

Indicate to the Feeder to start feeding the CEP engines with events coming from a topic.

Parameters:
`topic` - which the feeder must feed the CEP engine/s
Returns:
`true` if the feeding is possible. `false` otherwise.

5.3 Complex Event Handler

Like the Events Feeder, the CEH is an abstraction between the CEP-Engines and the action taken by the Handler. In this case the action of the fusion of events is a complex event which is published in the same manner as the events which triggered the complex event. If the way the events are published is changed or if the reaction to the events in the future should become more complex, like a rule engine, then the CEH could be changed without the need of rewriting all the different wrappers already developed.

As in the feeder, the CEH also unifies the logic of how the complex events are managed when a query is triggered.

5.3.1 Interface ComplexEventHandlerLogic

```
public interface ComplexEventHandlerLogic
```

One of the two interfaces which represent the API of the Complex Event Handler (CEH). The CEH is an abstraction between the CEP-Engine/s and the action taken by the Handler.
In this case the action of the fusion of events is a "complex event" which is published in the same manner as the events which trigger the complex event.
The CEH has two responsibilities. One is the action taken after the event was triggered (the handling of the event). The second is the interaction between the `DataFusionWrapper` and the event handler. This interface is the API of the first responsibility.

See Also:
`DataFusionWrapper`

- **Method Summary**

Modifier and Type	Method and Description
boolean	**addHandler**(java.lang.String name, java.lang.String query, java.lang.String topic) Add a handler to the requested query.

- **Method Detail**

1.5.2.1 addHandler

```
boolean addHandler(java.lang.String name, java.lang.String query,
java.lang.String topic)
```

Add a handler to the requested query.

Parameters:

`name` - of the query. This is how the query will be addressed.

Also name indicates where data is going to be published when the query is triggered e.g. "my.event.query" indicates that the event will be published in "/my/event/query/"

`query` - as String for the specific CEP engine (temporal)
`topic` - where the query will be deployed
Returns:
`true` if the query was deployed successfully. `false` otherwise.

5.3.2 Interface ComplexEventHandler

```
public interface ComplexEventHandler
```

One of the two Interfaces which represents the API of the Complex Event Handler (CEH). The CEH is anabstraction between the CEP-Engine/s and the action taken by the Handler.

In this case the action of the fusion of events is a "complex event" which is published in the same manner as the events which triggered the complex event.

The CEH has two responsibilities. One is the action taken after the event was triggered (the handling of the event). The second is the interaction between `DataFusionWrapper` and the event handler. This interface is the API of this second responsibility.

See Also:
`DataFusionWrapper`

- **Method Summary**

Modifier and Type	Method and Description Method and Description
boolean	**callerback**(ResponseSet answer) Function for send back the complex event to the complex event handler
boolean	**dataFusionWrapperSignIn**(DataFusionWrapper dfw) The CEH do not have an awareness of which engines are available.
boolean	**dataFusionWrapperSignOut**(DataFusionWrapper dfw) If a subscribed want to not be handled any more then have to indicated explicitly.

- **Method Detail**
  - **dataFusionWrapperSignIn**

```
boolean dataFusionWrapperSignIn(DataFusionWrapper dfw)
```

The CEH does not have an awareness of which engines are available.

For the Handler to be able to interact with a Data Fusion Engine, the wrapper of the engine has to explicitly subscribe to the handler as a Data Fusion engine, doing so through this function

Parameters:
`dfw` - is the `DataFusionWrapper` to be subscribed to.
Returns:
`true` in a successful subscription, `false` otherwise.

  - **dataFusionWrapperSignOut**

```
boolean dataFusionWrapperSignOut(DataFusionWrapper dfw)
```

If a subscriber does not want to be handled any more then this has to be indicated explicitly.

Parameters:
`dfw` - is the `DataFusionWrapper` to be unsubscribed to.
Returns:
`true` in a successful unsubscription, `false` otherwise.

  - **callerback**

```
boolean callerback(ResponseSet answer)
```

Function to send back the complex event to the complex event handler

Returns:
`true` if the event was handled successfully , `false` otherwise.

5.4 Data Fusion Engine Wrapper

The DF wrapper is an interface provided by the Data Fusion Manager. The interface provides an API by which the several engines connect to the IMPRESS framework. This API can be accessed natively in an OSGi environment or by Web Services allowing a highly decoupled infrastructure in which any engine could be added so long as a wrapper is implemented which interacts though Web Services.

5.4.1 Interface DataFusionWrapper

```
public interface DataFusionWrapper
```

The DF wrapper is an Interface. The Interface provides an API in which the several engines may be connected to the IMPReSS framework. This API can be accessed natively in an OSGi environment or by Web Services, allowing the a high decupled infrastructure in which any engine could be added so long as a wrapper is implemented which interacts through Web Services.

See Also:
`DataFusionWrapper`

- **Method Summary**

Modifier and Type	Method and Description
boolean	**addEvent**(java.lang.String topic, java.util.Map<java.lang.String,java.lang.Object> parts) Add send and event to the CEP engine
boolean	**addEventType**(java.lang.String nameType, java.lang.String[] eventSchema, java.lang.Object[] eventTypes) **Deprecated.**
boolean	**addQuery**(java.lang.String name, java.lang.String query, java.lang.String topic) Add a query to the CEP engine which would be later handler with a handler.
java.lang.String	**getName**() Return the name of the CEP which implements the interface
boolean	**setTopicToEventType**(java.lang.String topic, java.lang.String eventType) **Deprecated.**

- **Method Detail**
  - **getName**

    ```
    java.lang.String getName()
    ```

    Return the name of the CEP which implements the interface

    Returns:
    Name of the CEP engine wrapped

  - **addEvent**
  - `boolean addEvent(java.lang.String topic,`

    ```
    java.util.Map<java.lang.String,java.lang.Object> parts)
    ```

Add send and event to the CEP engine

Parameters:
`topic` - where the event was published
`parts` - is the event itself. A map of values and values names
Returns:
`true` if the event was added to the CEP engine. `false` otherwise.

- **addEventType**
- `@Deprecated`
- `boolean addEventType(java.lang.String nameType,`
- `java.lang.String[] eventSchema,`
  `java.lang.Object[] eventTypes)`

Deprecated.

Configure a particular type in the engine. This functionality is considered deprecated due to the unclearness of this concept as general concept for all `DataFusionWrappers`.

More research must be done whether to discard this feature or not.

Parameters:
`nameType` - are the names of type added in the engine
`eventSchema` - are the names of the inner values of the event
`eventTypes` - are the types of the inner values of the event
Returns:
`true` if the event was added to the CEP engine. `false` otherwise.

- **setTopicToEventType**
- `@Deprecated`
- `boolean setTopicToEventType(java.lang.String topic,`

`java.lang.String eventType)`

Deprecated.

Configure a particular type to a particular topic.

More research must be done whether to discard this feature or not.

Parameters:
`topic` - to be associated with a topic.
`eventType` - is the pre-configured in the CEP engine type which will be associated with a topic.
Returns:
`true` if the topic was configured to the CEP engine. `false` otherwise.

- **addQuery**
- `boolean addQuery(java.lang.String name,`
- `java.lang.String query,`
  `java.lang.String topic)`

Add a query to the CEP engine which would be later handler with a handler.

Parameters:
`name` - which references the query
`query` - to be implemented in the CEP engine.
`topic` - that the query will query for
Returns:
`true` if the query is successfully deployed in the CEP engine. `false` otherwise.

5.5 Other Classes

5.5.1 Interface ResponseSet

```
public interface ResponseSet
```

Response is an interface which identifies the functionality to access the resulting data of a SQL-like query generated by a Data Fusion Engine.

The interface represents an array of "rows" in a SQL-like response, this means a complete response of a SQL-like query.

The ResponseSet is a collection of Response.

The motivation of this interface is to allow the wrappers to implement the data structure as they please.

The functionality provided described in the interface allows the access to the data generated by a `DataFusionWrappers`. For the implementation of the interface an ArrayList or a List implementation is recommended.

See Also:
`DataFusionWrapper`, `ResponseSet.Response.Cell`, `ResponseSet.Response`

- **Nested Class Summary**

Nested Classes	
Modifier and Type	**Interface and Description**
static interface	**ResponseSet.Response** `Response` is an interface which identifies the functionality to access the resulting data of a SQL-like query generated by a Data Fusion Engine.

- **Method Summary**

Modifier and Type	Method and Description
ResponseSet.Response	**get**(int i) Get the selected `ResponseSet.Response` by parameter index.
java.lang.String	**getName**() Get the name of response set.
ResponseSet.Response[]	**getResponses**() Returns an array of `Response` containing all of the elements in this `ResponseSet` in proper sequence (from first to last element).
int	**size**() Returns the number of `Response`s in the `ResponseSet`

- **Method Detail**

- **getName**

```
java.lang.String getName()
```

Get the name of response set. This is the name of the query which generated the response.

This name is use later as the topic where the response will be published.

Returns:
The name of response set as String.

- **get**
- `ResponseSet.Response get(int i) throws java.lang.IndexOutOfBoundsException`

Get the selected `ResponseSet.Response` by parameter index.

Parameters:
`i` - is the index of the element to return
Returns:
The selected response.
Throws:
`java.lang.IndexOutOfBoundsException` - - if the index is out of range (index < 0 || index >= size())

- **getResponses**

```
ResponseSet.Response[] getResponses()
```

Returns an array of Response containing all of the elements in this ResponseSet in proper sequence (from first to last element).

Returns:
An array of Response containing all of the Responses.

- **size**

```
int size()
```

Returns the number of Responses in the ResponseSet.

Returns:
amount of elements contained in the ResponseSet.

5.5.2 Interface ResponseSet.Response

```
public static interface ResponseSet.Response
```

Response is an interface which identifies the functionality to access the resulting data of a SQL-like query generated by a Data Fusion Engine.
The interface represents a "row" in a SQL-like response.
The Response is a collections of Cells (see `Cell`).
The motivation of this interface is to allow the wrappers to implement the data structure as they please.

The functionality provided described in the interface allows access to the data generated by a `DataFusionWrapper`s. For the implementation of the interface a `Hashmap` or a `Map` implementation is recommended.

See Also:
`DataFusionWrapper`, `ResponseSet.Response.Cell`

- **Nested Class Summary**

Nested Classes	
Modifier and Type	**Interface and Description**
static interface	**ResponseSet.Response.Cell** Cell is an interface which identify the minimum functionality to access the resulting data of a SQL-like query generated by a Data Fusion Engine.

- **Method Summary**

Modifier and Type	Method and Description
boolean	**exist**(java.lang.String column) Check if a particular column name is contained in the Responses.
ResponseSet.Response.Cell	**getCell**(int columnNo) Obtain the `Cell` of the selected column name.
ResponseSet.Response.Cell[]	**getCells**() Get the cells contained in the `response`.
java.lang.String[]	**getColumnsNames**() Get the names of the each data in the data collection.
java.lang.Object[]	**getRawRow**() Get the inner values of the each data in the data collection.
java.lang.Object	**getValueOf**(java.lang.String columnName) Obtain the value of the selected column name.
int	**size**() Returns the number of `Cell`s in the `Response`

- **Method Detail**
  - **getColumnsNames**

    ```
    java.lang.String[] getColumnsNames()
    ```

    Get the name of the each data cell in the data collection. In the SQL semantics, the column names of the data.

    Returns:
    The names of the value as `String Array`.

  - **getRawRow**

    ```
    java.lang.Object[] getRawRow()
    ```

    Get the inner values of each data cell in the data collection. In the SQL semantics, the value contained in the cells.

    Returns:
    The values as Object Array.

  - **getCells**

```
ResponseSet.Response.Cell[] getCells()
```

Get the cells contained in the response. In the SQL semantics, the row with its correspondent column name

Returns:
The Cells as Array of `Cell`.
See Also:
`ResponseSet.Response.Cell`

- **exist**

```
boolean exist(java.lang.String column)
```

Check if a particular column name is contained in the responses.

Parameters:
`column` - is name of the column which will be queried.
Returns:
`true` if the name exist in the response, `false` otherwise.

- **getValueOf**

```
java.lang.Object getValueOf(java.lang.String columnName)
```

Obtain the value of the selected column name.

Parameters:
`columnName` - of the element to return.
Returns:
The value of the selected data by the parameter if exist, `null` otherwise.

- **getCell**

```
ResponseSet.Response.Cell getCell(int columnNo)
```

Obtain the `cell` of the selected column name.

Parameters:
`columnNo` - is the index of the element to return.
Returns:
The `Cell` of the selected column by the parameter if exist, `null` otherwise.
See Also:
`ResponseSet.Response.Cell`

- **size**

```
int size()
```

Returns the number of cells in the response.

Returns:
amount of elements contained in the response.

5.5.3 Interface ResponseSet.Response.Cell

```
public static interface ResponseSet.Response.Cell
```

Cell is an interface which identifies the minimum functionality to access the resulting data of a SQL-like query generated by a Data Fusion Engine. The interface represents where a "Column" and "row" cross SQL-like response.

The motivation of this interface is to allow the wrappers to implement the data structure as they please. The functionality provided described in the interface allows the access to the data generated by a `DataFusionWrapper`s.

See Also:
`DataFusionWrapper`

- **Method Summary**

Modifier and Type	Method and Description
`java.lang.String`	**`getName`**`()` Get the name of the data.
`java.lang.Object`	**`getValue`**`()` Get the value of the data.

- **Method Detail**
  - **getName**

    ```
    java.lang.String getName()
    ```

    Get the name of the data. In the SQL semantics the column name

    Returns:
    The name of the value as String.

  - **getValue**

    ```
    java.lang.Object getValue()
    ```

    Get the value of the data. In the SQL semantics the value contained in a cell (cross of a row with a column)

    Returns:
    The value as Object.

6. References

Adi, A., Botzer, D., Nechushtai, G., & Sharon, G. (2006). *Complex event processing for financial services.* Paper presented at the Services Computing Workshops, 2006. SCW'06. IEEE.

Baran, R. (2002). *A collective computation approach to automatic target recognition*.

Bedworth, M., & O'Brien, J. (2000). The Omnibus model: a new model of data fusion? *Aerospace and Electronic Systems Magazine, IEEE, 15*(4), 30-36.

Bracio, B. R., Horn, W., & Moller, D. P. F. (1997). *Sensor fusion in biomedical systems.* Paper presented at the Engineering in Medicine and Biology Society, 1997. Proceedings of the 19th Annual International Conference of the IEEE.

Brown, C., Durrant-Whyte, H., Leonard, J., Rao, B., & Steer, B. (1992). Distributed data fusion using Kalman filtering: A robotics application. In M. A. Abidi & R. C. Gonzalez (Eds.), *Data fusion in robotics and machine intelligence* (pp. 267–309): Academic Press Professional, Inc.

Cain, M., Stewart, S., & Morse, J. (1989). *Object classification using multispectral sensor data fusion*.

Candes, E. J., & Wakin, M. B. (2008). An Introduction To Compressive Sampling. *Signal Processing Magazine, IEEE, 25*(2), 21-30.

Chan Yet, W., & Qidwai, U. (2005). *Intelligent sensor network for obstacle avoidance strategy.* Paper presented at the Sensors, 2005 IEEE.

Ciancio, A., & Ortega, A. (2004). *A distributed wavelet compression algorithm for wireless sensor networks using lifting*.

Coates, M. (2004). *Distributed particle filters for sensor networks*.

Coue, C., Fraichard, T., Bessiere, P., & Mazer, E. (2002). *Multi-sensor data fusion using Bayesian programming: An automotive application.* Paper presented at the Intelligent Vehicle Symposium, 2002. IEEE.

Crisan, D., & Doucet, A. (2002). A survey of convergence results on particle filtering methods for practitioners. *Signal Processing, IEEE Transactions on, 50*(3), 736-746.

Cui, X., Hardin, T., Ragade, R. K., & Elmaghraby, A. S. (2004). *A swarm-based fuzzy logic control mobile sensor network for hazardous contaminants localization.* Paper presented at the Mobile Ad-hoc and Sensor Systems, 2004 IEEE International Conference on.

Dasarathy, B. V. (1997). Sensor fusion potential exploitation-innovative architectures and illustrative applications. *Proc. IEEE, 85*, 24-38.

Elmenreich, W. (2002). *Sensor fusion in time-triggered systems.* (Ph.D.), Vienna University of Technology, Vienna.

Ficco, M., & Romano, L. (2011). *A generic intrusion detection and diagnoser system based on complex event processing.* Paper presented at the Data Compression, Communications and Processing (CCP), 2011 First International Conference on.

Frankel, C. B., & Bedworth, M. D. (2000). *Control, estimation and abstraction in fusion architectures: lessons from human information processing.* Paper presented at the Information Fusion, 2000. FUSION 2000. Proceedings of the Third International Conference on.

Friedlander, D. (2005). Semantic information extraction. *Distributed Sensor Networks*, 409–417.

Gilks, W., Richardson, S., & Spiegelhalter, D. (1996). Markov Chain Monte Carlo in Practice. : Chapman & Hall.

Gupta, I., Riordan, D., & Srinivas, S. (2005). *Cluster-head election using fuzzy logic for wireless sensor networks.* Paper presented at the Communication Networks and Services Research Conference, 2005. Proceedings of the 3rd Annual.

Gustafsson, F., & Gunnarsson, F. (2003, 6-10 April 2003). *Positioning using time-difference of arrival measurements.* Paper presented at the Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on.

Halgamuge, M. N., Guru, S. M., & Jennings, A. (2003). *Energy efficient cluster formation in wireless sensor networks.* Paper presented at the Telecommunications, 2003. ICT 2003. 10th International Conference on.

Hongyang, C., Deng, P., Xu, Y., & Li, X. (2005, 23-26 Sept. 2005). *A robust location algorithm with biased extended Kalman filtering of TDOA data for wireless sensor networks.* Paper presented at the Wireless Communications, Networking and Mobile Computing, 2005. Proceedings. 2005 International Conference on.

Hu, L., & Evans, D. (2004). *Localization for mobile sensor networks*.

Intanagonwiwat, C., Govindan, R., & Estrin, D. (2000). *Directed diffusion: a scalable and robust communication paradigm for sensor networks*. Paper presented at the Proceedings of the 6th annual international conference on Mobile computing and networking, Boston, Massachusetts, United States.

Isard, M., & Blake, A. (1996). Contour tracking by stochastic propagation of conditional density. In B. Buxton & R. Cipolla (Eds.), *Computer Vision — ECCV '96* (Vol. 1064, pp. 343-356): Springer Berlin / Heidelberg.

Iyengar, S. S., Chakrabarty, K., & QI, H. (2001). Introduction to special issue on “distributed sensor networks for real-time systems with adaptive configuration". *Journal of Franklin Institute, 338*, 655-668, 651-653.

Kokar, M. M., Bedworth, M. D., & Frankel, C. B. (2000). *Reference model for data fusion systems*.

Lei, F., Wenliang, D., & Peng, N. (2005). *A beacon-less location discovery scheme for wireless sensor networks.* Paper presented at the INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE.

Lewis, T. W., & Powers, D. M. W. (2002). *Audio-visual speech recognition using red exclusion and neural networks*. Paper presented at the Proceedings of the twenty-fifth Australasian conference on Computer science - Volume 4, Melbourne, Victoria, Australia.

Li, S., Son, S. H., & Stankovic, J. A. (2003). *Event detection services using data service middleware in distributed sensor networks*. Paper presented at the Proceedings of the 2nd international conference on Information processing in sensor networks, Palo Alto, CA, USA.

Li, T., Ekpenyong, A., & Yih-Fang, H. (2006). Source Localization and Tracking Using Distributed Asynchronous Sensors. *Signal Processing, IEEE Transactions on, 54*(10), 3991-4003.

Liggins, M. E., Hall, D. L., & Llinas, J. (2009). *Handbook of Multisensor Data fusion, Theory and Practice*. Boca Raton, FL: CRC Press.

Liu, J., & Zhao, F. (2006). *Towards semantic services for sensor-rich information systems*.

Luo, R. C., Chih-Chen, Y., & Kuo Lan, S. (2002). Multisensor fusion and integration: approaches, applications, and future research directions. *Sensors Journal, IEEE, 2*(2), 107-119.

Madden, S. R., Franklin, M. J., Hellerstein, J. M., & Hong, W. (2005). TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst., 30*(1), 122-173. doi: http://doi.acm.org/10.1145/1061318.1061322

Mascolo, C., & Musolesi, M. (2006). *SCAR: context-aware adaptive routing in delay tolerant mobile sensor networks*. Paper presented at the Proceedings of the 2006 international conference on Wireless communications and mobile computing, Vancouver, British Columbia, Canada.

Miguez, J., & Artes-Rodriguez, A. (2006). *A Monte Carlo method for joint node location and maneuvering target tracking in a sensor network*.

Nakamura, E., Nakamura, F., Figueiredo, C., & Loureiro, A. (2005). Using Information Fusion to Assist Data Dissemination in Wireless Sensor Networks. *Telecommunication Systems, 30*, 237-254.

Nakamura, E. F., Loureiro, A. A. F., & Frery, A. C. (2007). Information fusion for wireless sensor networks: Methods, models, and classifications. *ACM Comput. Surv., 39*(3), 9. doi: http://doi.acm.org/10.1145/1267070.1267073

Patwari, N., Hero, A. O., III, Perkins, M., Correal, N. S., & O'Dea, R. J. (2003). Relative location estimation in wireless sensor networks. *Signal Processing, IEEE Transactions on, 51*(8), 2137-2148.

Rabbat, M., & Nowak, R. (2004). *Distributed optimization in sensor networks*. Paper presented at the Proceedings of the 3rd international symposium on Information processing in sensor networks, Berkeley, California, USA.

Rachlin, Y., Negi, R., & Khosla, P. (2006). *On the interdependence of sensing and estimation complexity in sensor networks*. Paper presented at the Proceedings of the 5th international conference on Information processing in sensor networks, Nashville, Tennessee, USA.

Sam, D., Nwankpa, C., & Niebur, D. (2001). *Decision fusion of voltage stability indicators for small sized power systems.* Paper presented at the Power Engineering Society Summer Meeting, 2001. IEEE.

Savvides, A., Park, H., & Srivastava, M. B. (2003). The n-hop multilateration primitive for node localization problems. *Mob. Netw. Appl., 8*(4), 443-451. doi: http://dx.doi.org/10.1023/A:1024544032357

Schmitt, T., Hanek, R., Buck, S., & Beetz, M. (2001). *Cooperative probabilistic state estimation for vision-based autonomous mobile robots.* Paper presented at the Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on.

Sheng, X., Hu, Y., & Ramanathan, P. (2005). *Distributed particle filter with GMM approximation for multiple targets localization and tracking in wireless sensor network*.

Singh, A., Nowak, R., & Ramanathan, P. (2006). *Active learning for adaptive mobile sensing networks*. Paper presented at the Proceedings of the 5th international conference on Information processing in sensor networks, Nashville, Tennessee, USA.

Tang, C., & Raghavendra, C. (2006). *Wavelet based source broadcast for in-network processing in sensor networks unknown side information*.

Wei, W. W.-S. (1994). *Time series analysis*: Addison-Wesley publ.

White, F. E. (1991). Data fusion lexicon *Technical Panel for C3*. San Diego, CA: U.S. Department of Defense.

Willett, R., Martin, A., & Nowak, R. (2004). *Backcasting: adaptive sampling for sensor networks*. Paper presented at the Proceedings of the 3rd international symposium on Information processing in sensor networks, Berkeley, California, USA.

Wong, Y., Wu, J., Ngoh, L., & Wong, W. (2004). *Collaborative data fusion tracking in sensor networks using monte carlo methods*.

Xiao, L., Boyd, S., & Lall, S. (2005). *A scheme for robust distributed sensor fusion based on average consensus*. Paper presented at the Proceedings of the 4th international symposium on Information processing in sensor networks, Los Angeles, California.

Yiyao, L., Venkatesh, Y. V., & Ko, C. C. (2001). A knowledge-based neural network for fusing edge maps of multi-sensor images. *Information Fusion, 2*(2), 121-133.

Yu, B., Sycara, K., Giampapa, J., & Owens, S. (2004). *Uncertain information fusion for force aggregation and classification in airborne sensor networks*.