D1.2 Integrated Component Prototype

ABSTRACT

This deliverable produces the module prototypes of the components described in D1.1 as well as any interconnection modules with the TagItSmart platform, together with detailed documentation on how to install, configure, setup and extend the proposed implementation. To this end it utilizes the design of T1.1 and proceeds with the implementation of the extensions as well as the integration with the TagItSmart environment, primarily EVRYTHNG platform. It also implements the necessary logic in order to address experimentation scenarios, while incorporating specifications regarding deployment, installation and configuration details for the components. This report aims at clarifying these aspects and serve as the documentation and manual for these components.

Delivery Date: 31.01.2018

Work Package: 1

Task: T1.2

Dissemination Level: Public

Type of Deliverable:
Prototype

Lead partner: ELKE/HUA

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

CONTRIBUTORS

	ELKE/HUA
Authors	George Kousiouris, Stylianos Tsarsitalidis, Stavros Koloniaris, Evangelos Psomakelis, Konstantinos Tserpes, Dimosthenis Anagnostopoulos

DOCUMENT LOG

Issue	Date	Comment	Author/Partner
0.1	17/11/2017	Table of contents	George Kousiouris
0.2	10/12/2017	Incorporation of section for link to EVRYTHNG roles and API	George Kousiouris
0.3	15/12/2017	Updated ontological concepts table	George Kousiouris, Stylianos Tsarsitalidis
	23/12/2017	Updated scenario on weather incorporation	Vangelis Psomakelis
0.3.1	05/01/2018	Included system setup and configuration	Stavros Koloniaris
0.3.2	10/01/2018	Inclusion of UI elements	George Kousiouris, Stylianos Tsarsitalidis
0.3.3	15/01/2018	Inclusion of KB system backend	Stylianos Tsarsitalidis
0.3.4	20/01/2018	Inclusion of modelling framework and relevant flows	Vangelis Psomakelis
0.4	22/01/2018	Inclusion of helper flows	Stylianos Tsarsitalidis
0.5	31/01/2018	Inclusion of Exec summary, introduction, conclusions, merging of sections	George Kousiouris
1.0	03/02/2018	Final version ready	All

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

ACRONYMS

Acronym	Description
API	Application Programming Interface
HTTP	Hyper Text Transfer Protocol
KB	Knowledge Base
LCC	Large Crowd Concentration
NE	North East
OATH	Open Authentication
OWL	Ontology Web Language
RDF	Resource Description Framework
REST	Representational State Transfer
ST	Smart Tag
SW	South West
TIS	Tag It Smart
UC	Use Case
UI	User Interface
UML	Unified Modelling Language
URL	Uniform Resource Locator
VE	Virtual Entity

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

Contents

Executive Summary 9
1 Introduction 10
1.1 Implementation Tasks breakdown 10
2 External Inputs and Feedback 12
2.1 Needed events definition through co-creation 12
2.2 External (to AffectUs) Dependencies 12
2.2.1 External Data sources description 13
2.2.2 Link with TIS and incorporation of review feedback from D1.1 16
3 System Description and Installation Instructions 20
3.1 Complete System diagram 20
3.2 Integrated Packaging 21
3.3 System Setup Description 23
3.3.1 Single Step Deployment Process 27
3.4 License and Contact 31
4 Component Description and Manual 32
4.1 UI Elements 32
4.1.1 General UI Setup 32
4.1.2 Product StakeholderUI (Enable Access and Declare Dependencies) 33
4.1.3 Product Stakeholder UI (Chain History) 38
4.1.4 App Developer UI (Declare Chain) 40
4.1.5 App Developer UI (Monitor My Chain) 44
4.1.6 External Analytics Developers UI (Declare Provided Events) 46
4.2 Semantic framework and service 49
4.2.1 Finalized semantic structure and OWL 49
4.2.2 Semantic Framework structure and implementation 56
4.3 Prediction Framework 60
4.3.1 Description of available prediction methods 60
4.3.2 Prediction Framework structure and implementation 61
4.3.3 Usage 64

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

4.3.4 Future Work (Model Creation UI) 65
4.4 Linking/Integration Runtime layer 65
4.4.1 Scan data ingestion and transformation 65
4.4.2 External Event Publication Template definition 66
4.4.3 Specific Event detection logic (Abnormal Sequence of States) 67
4.4.4 Messaging Logic 68
5 Conclusions 71
6 References 74

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

INDEX OF TABLES

Table 1: List of software elements used in the AffectUs system 21
Table 2: List of external Docker images used 23
Table 3: Semantic concepts included in the AffectUs Domain 50
Table 4: Updated Requirements Traceability Matrix Following Implementation and Mapping to Flows . 71

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

INDEX OF FIGURES

Figure 1: Weather Data Node-RED flow. 13
Figure 2: JSON schema of produced weather events 14
Figure 3: Weather Interface 14
Figure 4: JSON example of LCC event publication 15
Figure 5: Traffic Node-RED. 15
Figure 6:Traffic User Interface. 16
Figure 7: JSON example of traffic event publication 16
Figure 8: The developers activities (green rectangle) and the AffectUS procedures (red rectangle) 20
Figure 9: Positioning of AffectUs system 21
Figure 10: A graphical representation of the running services, given by the visualization service 27
Figure 11: Single step deployment descriptor (Docker Service YAML file) for the AffectUs system 30
Figure 12: Top Level Main Menu Setup 32
Figure 13: Side Navigation Panel of UI 32
Figure 14: Generic Role interaction with the AffectUs system (Product Stakeholder UI coverage) 33
Figure 15: Product Owner Enable Access UI 34
Figure 16: Node-RED implementation of Enable Access UI 36
Figure 17: Product format coming from TIS EVRYTHNG 37
Figure 18: Chain History UI in the general use case 38
Figure 19: Chain History Node-RED flow. 39
Figure 20: Chain History User Interface. 39
Figure 21: JSON example of retrieved historical chain status from MongoDB 40
Figure 22: Generic Role interaction with the AffectUs system (App Developer UI coverage) 41
Figure 23: App Developer Declare Chain UI 42
Figure 24: Node-RED flow for implanting the App Dev Declare Chain functionality 43
Figure 25: Chain monitor UI in the general use case. 44
Figure 26: Chain Monitor Node-RED flow 45
Figure 27: Chain Monitor Interface. 45
Figure 28: JSON example of Chain Monitor 46
Figure 29: External Analytics Developer UI 47
Figure 30: External Analytics Developer Flow 48
Figure 31: JSON example of States Declaration 49
Figure 32: Ontology Part 1: EventCondition and Related Classes 54
Figure 33: Ontology Part 2: Supply Chain and Product 55
Figure 34: Ontology Part 3: Supply Chain and the rest 55
Figure 35: Example of Linked Data usage in AffectUs 56
Figure 36: Jena Fuseki Server Configuration 58
Figure 37: The Event Condition States schema for the validator 60

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

Figure 38: Prediction Modelling Node-RED flow 62
Figure 39: JSON example of model input including external events 62
Figure 40: Example of JSON publication for supply chain related event detection 63
Figure 41: Chain Model place in the general use case diagram 64
Figure 42: Scan data ingestion flow 66
Figure 43: JSON example of event publication 66
Figure 44: Abnormal Sequence flow 67
Figure 45: Structure of the Notification Abstraction Layer (Events_2_KB_flow) 69

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

Executive Summary

AffectUs is an extension module for the TIS platform that aims to cover a set of functionalities aiming to

- Detect abnormal conditions with relation to the state of a supply chain, based on models created from historical data regarding transition times from stage to stage
- Link and forward externally identified generic events (coming from open data sources, smart city data and external developers) to affected entities of the chain
- Increase link between verticals through identifying and implementing links between generic events and effects on a given Thing/product
- Give the ability to involved entities to abstractedly declare necessary concepts, structures and relationships

Through the joint usage of semantics, events identification and forwarding, AffectUs may enable this cross-fertilization of event notifications between verticals, thus enriching application context. Automated discovery of what products are affected by the predicted events is part of this process.

In this document, the implementation of the components following their design from D1.1 is presented, along with usage and configuration details. Initially the scope and objectives of the document are described in the Introduction, including reusability from external audiences and coverage of the requirements expressed in D1.1, while the incorporated external data sources are included in Section 2, along with the corrections made in scope and implementation following the feedback from D1.1.

The overall system setup is included in Section 3, including configuration and setup details for the dockerized version of the tools, which will aid in the fastest and seamless integration and usage of the provided software.

Section 4 contains the implementation and usage description of each individual element of the system, thus referring to externally facing user interfaces, information templates, needed information in each field and relation to the other elements of the system (such as database back-ends, knowledge base back-ends, EVRYTHNG platform etc.). Information is also included on the setup, implementation and operation of these elements, as well as updates where needed with relation to the initially defined concepts in D1.1 (such as the ontology concepts update in Section 4.1 and the location of implementation of the relevant triple creation). Structure and rationale of the supply chain modelling and prediction framework is included in a generic manner, following its declaration through the UIs, exploiting data streams coming from TIS and mapping them to the modelled supply chain concepts. A set of other needed runtime flows is described, operating as integration flows between the various system elements and transforming information where necessary.

Finally in Section 5, a mapping is performed on the Requirements Traceability Matrix stemming from D1.1 in order to validate that the overall system functionality has been covered by the developments detailed in this document, resulting in a validation of the extension completeness with relation to the original design.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

1 Introduction

The aim of this document is to summarize and describe the implementation work performed in the context of the AffectUs project extension, starting in parallel to the design stage of the project, but culminating just after the design's finalization from M2 and on. The objective of the implementation stage is to exploit the outcomes of the design phase and fulfill the requirements posed by the latter.

Overall, 6 requirements have been identified in D1.1, along with 5 system use cases and relevant sequence diagrams. The major areas of implementation include the front-end, the semantic backend, the modelling backend and the runtime integration layer with TIS/EVRYTHNG. For these subsystems the relevant information is provided on implementation details, configuration and usage aspects. The detailed tasks breakdown for each part is included in Section 1.1.

1.1 Implementation Tasks breakdown

In terms of tasks breakdown (in an Agile-like manner), the following main ones have been identified and followed:

- Implementation of AffectUs Association and Communication logic
  - Transferring of the semantic concepts as OWL relationships
  - Sanity check and avoidance of overlapping definitions
  - Investigation of means of inserting the triples in the KB and definition of the way (e.g. through user directed input, through messaging transformation etc.)
  - Investigation of dependencies with other elements (e.g. supply chain models)
  - Implementation of the respective semantic queries defined during the design stage and relevant clients to accept these as parametric arguments
  - Implementation of link between query responses and messaging system configuration/registration/acquisition of information
    - Includes integration to the main TIS platform messaging system and exploits the semantic links indicating the effect between Things. This effect should be translated to the technical link in the messaging system as follows
      - Data receipt->semantic query to detect dependencies-> alert affected entities
      - Data receipt-> translation to chain stage-> alert of model element

- Prediction framework implementation
  - Data inputs ingestion and transformation
    - Adaptation to specific formats from external data sources and from TIS as well as necessary actions (e.g. preprocessing, normalization etc.)
  - Parametric method creation and adaptation to input specification
  - Link with semantic layer to indicate which model identifies which event
  - Prediction model runtime usage

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

    - Receipt of current values and model estimation
    - Semantic query implementation for affected endpoints to be notified
    - Notification implementation between the current model estimation and the respective endpoint

- Implementation of user facing interfaces
    - Collect and map requirements to functionalities
    - Implement functionalities and produce output to respective dependent elements (KB, internal DB etc.)
    - Test usability and completeness of information inserted

- Packaging of system elements
    - Division of elements per Docker image and inclusion of dependencies
    - Image creation from configured container
    - System design in terms of service and network setup
        - Includes creation of the relevant service description yml file
    - Deployment of services

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

2 External Inputs and Feedback

2.1 Needed events definition through co-creation

One of the key aspects of the extension is the ability to combine different data sources in a manner that will concentrate and generate more knowledge and proactive management on behalf of the stakeholders in the system. As identified in D1.1, the following list of candidate events has been compiled:

- Events that have to do with delays in the transition between stages of the supply chain and can be detected through the exploitation of historical data (e.g. for the extraction of average times or other statistics characteristic of a transition).
- Abnormal events in the sense of an unfeasible or illegal sequence of appearance, for example:
    - scanned twice at selling point
    - scanned outside a designated region of sale
    - scanned at recycling point while not scanned at selling point
    - scanned at recycling point and then resold at selling point
    - scanned at selling point while scanned in the past and found violation in the temperature threshold. This event also captures the requirement for using the internal sensor values of the tags
- Batch recall for an event coming in related to one of the parts that indicates defective batch, needs to be associated to the selling points of that batch in order to speed up the recall process.
- Relevant notifications should be issued for missed scans based on the normal chain sequence.

External data sources that may be used include:

- Social network data usage, in order to detect large crowd concentrations in a given area, that might affect a number of aspects such as:
    - Increased consumption of the given product
    - Increased delays in the transportation phase that include the locations around which the event has been detected
- Weather data that can affect:
    - Transport times
    - Production especially of agricultural products that may be used as raw material in the production of a TIS product

2.2 External (to AffectUs) Dependencies

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

2.2.1 External Data sources description

2.2.1.1 Weather Events

The Weather data source is based on two open APIs, the DarkSky and the aWhere developer API. The Node-RED flow developed and presented in the following figure is receiving requests, containing a time, location and window size. Then it creates a weather norms array, by averaging the weather features of the days before and after the time provided, for the last ten years, by using the aWhere API. The number of days to be used for this average is set by the window input parameter. After that, an http request is done to the DarkSky API, requesting the weather information for the time provided. This information is then compared to the norms extracted and a number of events are produced and forwarded as a response to the original request.

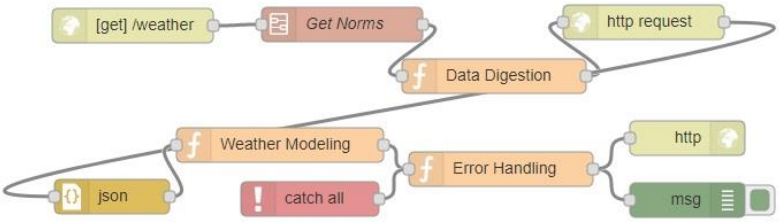


Figure 1: Weather Data Node-RED flow.

The events supported at the time are related to the temperature, the wind levels and the relative humidity as follows:

```
{
 "eventConditionname":"temperature",
 "statename": "cold",
 "Details": {"timestamp": value,"location": {GEOJSON},"value":value}
}
{
"eventConditionname":"temperature",
 "statename": "hot",
 "Details": {"timestamp": value,"location": {GEOJSON},"value":value}
}
{
 "eventConditionname":"wind",
 "statename": "windy",
 "Details": {"timestamp": value,"location": {GEOJSON},"value":value}
}
{
"eventConditionname":"humidity",
 "statename": "humid",
 "Details": {"timestamp": value,"location": {GEOJSON},"value":value}
```

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

```
}
{
 “eventConditionname”:”humidity”,
 “statename”: “dry”,
 “Details“: {“timestamp”: value,“location”: {GEOJSON},”value”:value}
}
```

Figure 2: JSON schema of produced weather events

In addition to that, we also have developed a User Interface that enables a user to explore the weather data, fiddling with the parameters in real time. An overview of the UI is presented in the following figure:

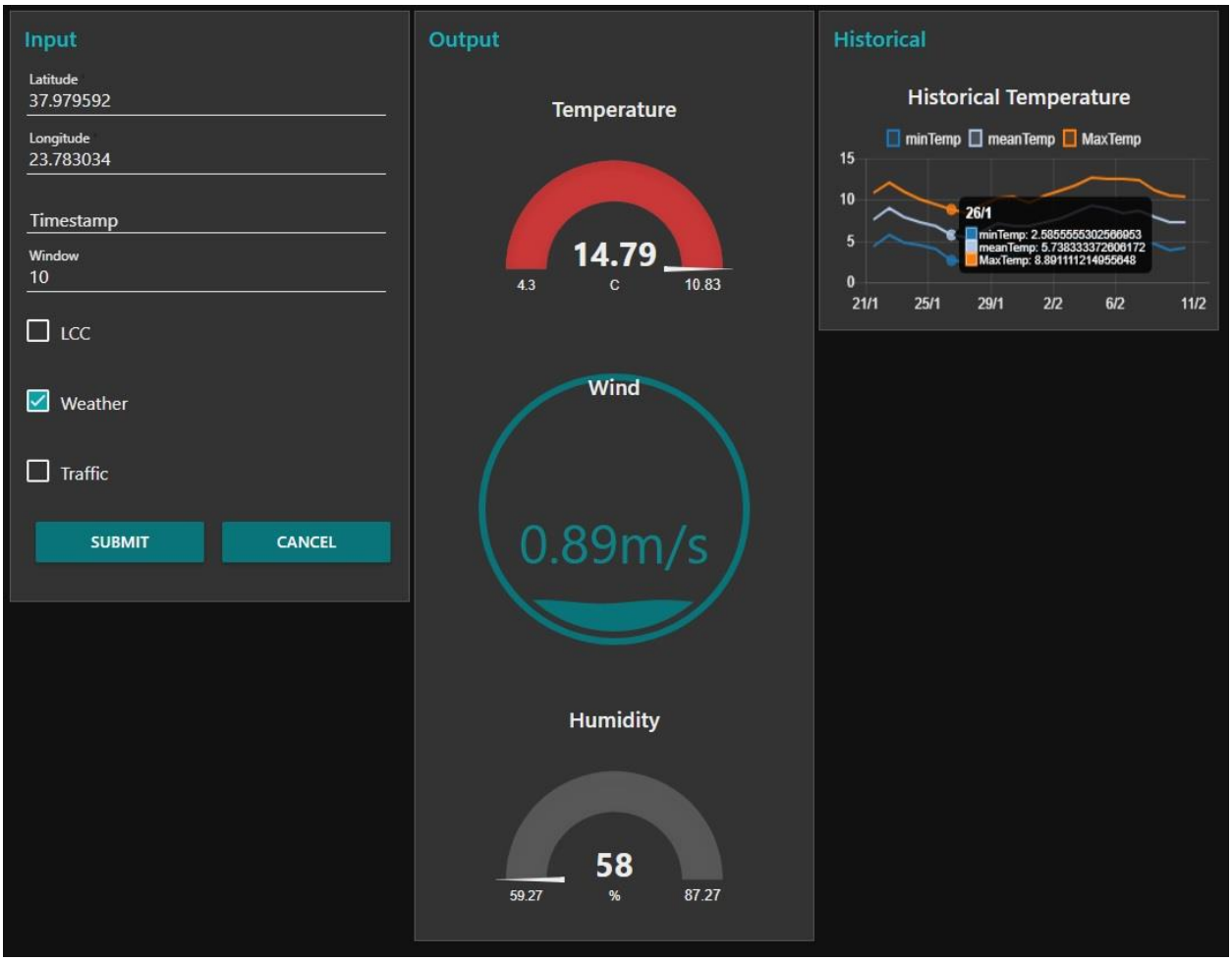


Figure 3: Weather Interface

As seen in the figure, the weather UI is expecting a geolocation coordinates input as well as a timestamp in UNIX millisecond epoch format and a window size in order to provide the relevant information. Then it fetches the temperature, wind and humidity readings and presents them in a comparative format to the historical norms of the last ten years. This is accomplished by using the gauges, where the minimum reading (left) is the historical minimum norm, the maximum reading (right) is the historical maximum

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

norm and the central reading is the current condition. In the chart to the right we can see how the norms progress inside the window we defined with the window parameter. Finally, the wind is presented as a rising level, with maximum being the historical maximum norm and minimum being the value of 0m/s.

2.2.1.2 LCC event definition

For the LCC event case we adapted the flow created in the context of the COSMOS project ([3], [10]) in order to adapt to the output template needed (Figure 4), as indicated in Section 4.4.2.

```
{
 "eventConditionname":"CrowdConcentration",
"statename": "High", //or "Low"
"Details": {
        "timestamp": value,"location": {GEOJSON},
        "tweetCount":value,
        "threshold":value
        }
}
```

Figure 4: JSON example of LCC event publication

2.2.1.3 Traffic State

The Traffic data source is using the Google Transit API in order to identify the traffic levels in the area, or route requested. The request comes either from a RestFul web service API or from the User Interface developed for the Traffic data source. In the first case, the user needs to provide a JSON object as an input, containing the timestamp of interest in UNIX millisecond epoch format and either a bounding box or a route, in order to define the area of interest. The Bounding Box needs to be in a JSON format containing the top-right coordinates of the map and the bottom-left coordinates in order to form the box. If the user chose the route, she needs to provide a JSON array, containing the points of interest along the route, which are of start, turn and stop types. This also enables the user to define a route with intermediate stops along the route, useful for product deliveries. An overview of the Node-RED flow is presented here:

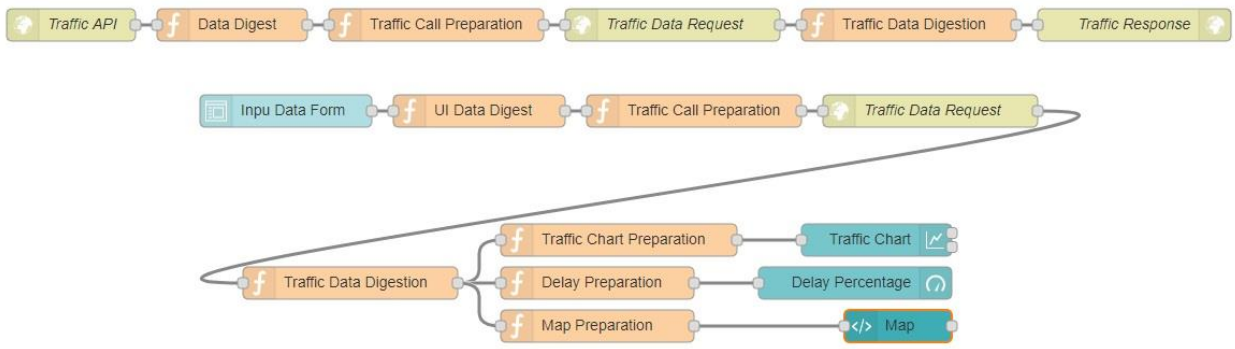


Figure 5: Traffic Node-RED.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

The interface presents the data in the form of a traffic chart, containing information about the traffic through the day, a delay percentage gauge that shows the estimated delay and a map that shows the traffic conditions in the area of interest. An overview of this interface is presented here:

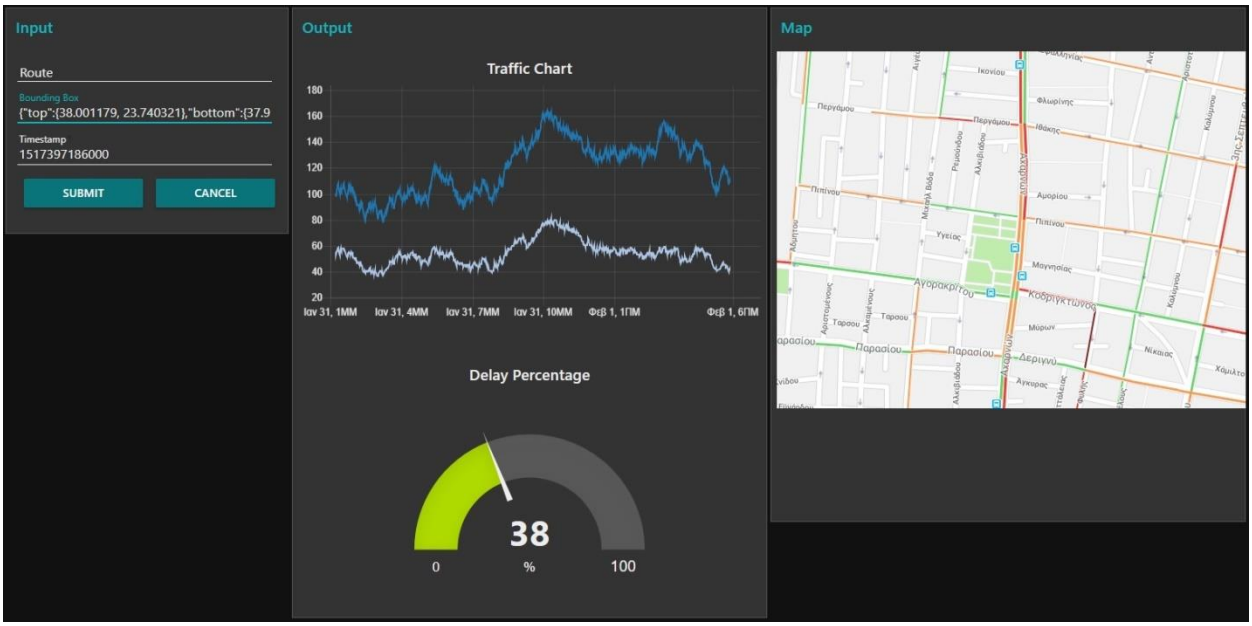


Figure 6:Traffic User Interface.

The Traffic data source flow is also creating an event in the case of high traffic during the shipping of tracked products. The event has the following structure:

```
{
    “eventConditionname”:”traffic”,
 “statename”: “high_traffic”,
 “Details“: {
 “timestamp”: value,
 “bounding_box”: {“top”:GEOJSON,”bottom”:GEOJSON},
 ”delay_percentage”:value}
```

Figure 7: JSON example of traffic event publication

2.2.2 Link with TIS and incorporation of review feedback from D1.1

The link with the EVRYTHNG platform is the main integration point of AffectUs with TIS. From an AffectUs point of view, what is initially needed is an Application Role (and respective API key) inside the project of a given EVRYTHNG account, that holds the information about products and things. Using these credentials, AffectUs may act as a bridge between affected products, based on the declaration of one of the product owners. Obviously the other involved product owner needs also to agree and provide this application level role to the AffectUs entity acting as the bridge.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

One of the major benefits of AffectUs is the fact that it transcends the boundaries of an EVRYTHNG account and can link between seemingly disjoint conditions that are indirectly declared. While in EVRYTHNG there is the possibility to give application role to an external entity (e.g. consumer of your products) in order to check the conditions or insert rules, this does not propagate further (e.g. to the consumer of your consumer), thus limiting to early warnings only with relation to the previous step. With AffectUs this limitation does not exist, given that we can reach sequentially from one affected entity to the next without limitation in the number of levels included, thus detecting earlier an abnormal condition, while including also the aspects of external events inclusion.

2.2.2.1 Terminology considerations

In the review of D1.1, both reviewers commented on the need to include TIS terminology in the rationale of AffectUs. For this reason we have tried throughout this document to implement this consideration (e.g. replace the EVRYTHNG product concept to the VE etc.).

2.2.2.2 Supply Chain Focus related to VE Lifecycle Management

In the review of D1.1 it was mentioned that we should focus primarily on the VE lifecycle as a more generic concept rather than a supply chain scope which can be a specific case for Lifecycle Management.

In order to address this comment, we have broadened the examined scope and have reformulated the concept of the supply chain stages to include types of lifecycle management. For example the ontological concept of StageType revolves around lifecycle management, including stages related to the VE lifecycle such as out of production, in transit, at retail, at end user and at recycle bin, and the chain model also is built around these types. Furthermore the supply chain focus can be strengthened or weakened depending on whether the interested party focuses on e.g. the VE (product) level or the collection (product batch) level, a feature feasible to be toggled initially through the EVRYTHNG platform and secondary in AffectUs by adapting the respective registration HTTP calls to the pub/sub system (e.g. /collections versus /products/productID level).

2.2.2.3 Description of the techniques, the data and the integration with TIS

In the review of D1.1 it was mentioned that a clear description of the techniques, the data and the integration with TIS should be included, as well as the links between Semantics and e.g. Machine learning from the back end, links to functionalities etc.

For this reason we have adapted the structure of Section 4 to address these issues and include:

- Link between the user interfaces and functionalities
- Link between the UI functionalities and related integration points with the Semantic back end, the ML back end and the TIS framework
- Link between Semantics and Modelling in order to translate TIS incoming data to annotated and semantically enriched notifications
- Inclusion of an “Information template” subsection to indicate dependencies on data and relevant structures

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

2.2.2.4 Scalability of the involved methods

In the review of D1.1 it was mentioned that scalability of the provided framework should be discussed. In this context, much of the operations in the AffectUs extension relate to correlation and setup actions that trigger the respective configuration in the components needing more scalability features, such as the messaging system. The configurations themselves (e.g. dictating the necessary bindings between messaging topics based on affected entities) do not have high scalability requirements, while messaging has been based on popular and scalable solutions such as RabbitMQ.

Increased scalability can be mitigated given the dockerization of AffectUs, thus giving us the ability to follow aspects such as launching individual containers per entity for certain elements that present these scalability challenges (such as Node-RED). The considered major bottleneck refers to the receipt and translation of the scan data to the chainstage type, which involves querying the KB for correlating location to stage type (included in Section 4.4.1). For this case a replication strategy of the KB could be followed in order to have an instance per chain. For some of the system elements alternative implementations have been drafted in order to enhance scalability aspects (e.g. Section 0 on the messaging structure).

2.2.2.5 Data Requirements from TIS/EVRYTHNG

In the review of D1.1 it was mentioned that data requirements from TIS in order to train the models could be provided.

In order to address this comment, specific guidelines and data formats have been included in each section where applicable, especially in the case of declarations performed by the various entities in AffectUs (either coming from TIS as product owners, e.g. Sections 4.1.2 for product data, 4.1.4 for the VE lifecycle chain modelling or coming from external developers providing generic event notifications and the related publication schemas in sections 4.1.6 and 4.4.2). Furthermore, access rights for the needed AffectUs application role on a product's account are included in Section 4.1.2.

With relation to the requirements expressed in D1.1, the existence of unique IDs was validated and exists in TIS/EVRYTHNG (product ID and Thng ID) while it is further enhanced from our baseline requirement given that EVRYTHNG supports also the grouping of Thngs into collections, which can be a very useful feature for future work in e.g. batch recall approaches.

With relation to concrete usage of information from TIS, the following data elements/features are considered necessary:

- Scan data coming in from the registered VEs and the respective Thngs, by connecting to EVRYTHNG pub/sub MQTT system. Action rules can also be used in order to act as a filter for reducing information received or for directly including the description of a chainstage based on a given location. This can act in parallel to the internal AffectUs mapping from location to stage type for e.g. enhancing scalability aspects. If the respective right is assigned, the AffectUs flow could also translate the initial VE lifecycle stage declarations into HTTP calls towards EVRYTHNG to create the respective actions or declare the relevant Places concept in EVRYTHNG.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

- Pushing of notifications from AffectUs to the product owner could be directly performed by publishing information on a related EVRYTHNG MQTT topic declared by the latter
- OATH based application developer role from the EVRYTHNG product owner to AffectUs should be assigned (details on section 4.1.2). Only the API key is needed as input, following a specific process of creating it in the EVRYTHNG platform.
- Receipt of VE list should be performed by AffectUs in order to give the VE owner the ability to populate needed declarations in an easier manner

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

3 System Description and Installation Instructions

3.1 Complete System diagram

The purpose of AffectUs is to create a way of watching over and controlling the whole procedure of the life cycle and supply chain of a given product, from the early stages of primary production till the end of the product's life, whether this is its destruction or recycling. This is achieved by using the technology of TIS in order to pinpoint a product's location and its current state in the levels of production, handling and disposal. The smart tags, scanned in each of the aforementioned stages will provide the necessary information about the product's location, expiration date, current temperature etc. This information will be forwarded to AffectUs for processing and creation of the needed notifications of state. The reports in their turn will be forwarded to the production managers and stakeholders in order to assist them in taking the best decisions and making the best planning in order to optimize their business.

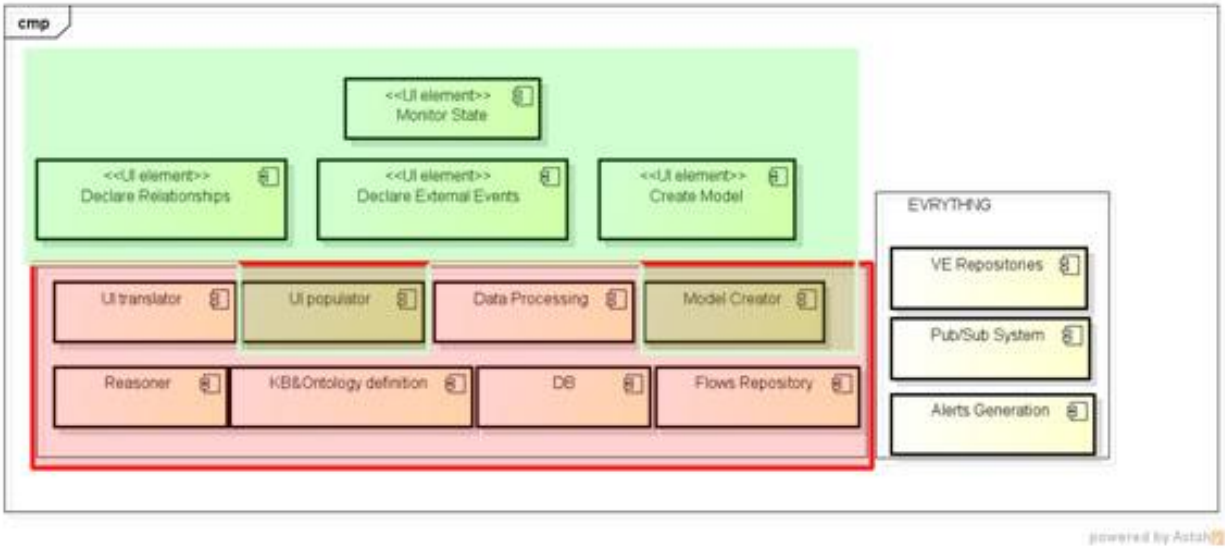


Figure 8: The developers activities (green rectangle) and the AffectUS procedures (red rectangle)

Firstly, an external entity(developer, product owner etc.)is needed to dictate the conditions under which the AffectUs platform would make the processing of the data and create the reports to be forwarded. Thus, a UI would exist that will have all the necessary tools and options needed for the entity to import the data, create the requested processes, declare the events that need to be traced and monitored, create relationships between events and start up the monitoring procedure. The TIS platform will forward all receiving information from a product's smart tag, while external sources (e.g. Weather conditions) are also forwarded to the AffectUs platform from external analytics developers. This information would be processed according to the operations requested and developed in order to create the needed reports. The reports would then be sent out to the stakeholders.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

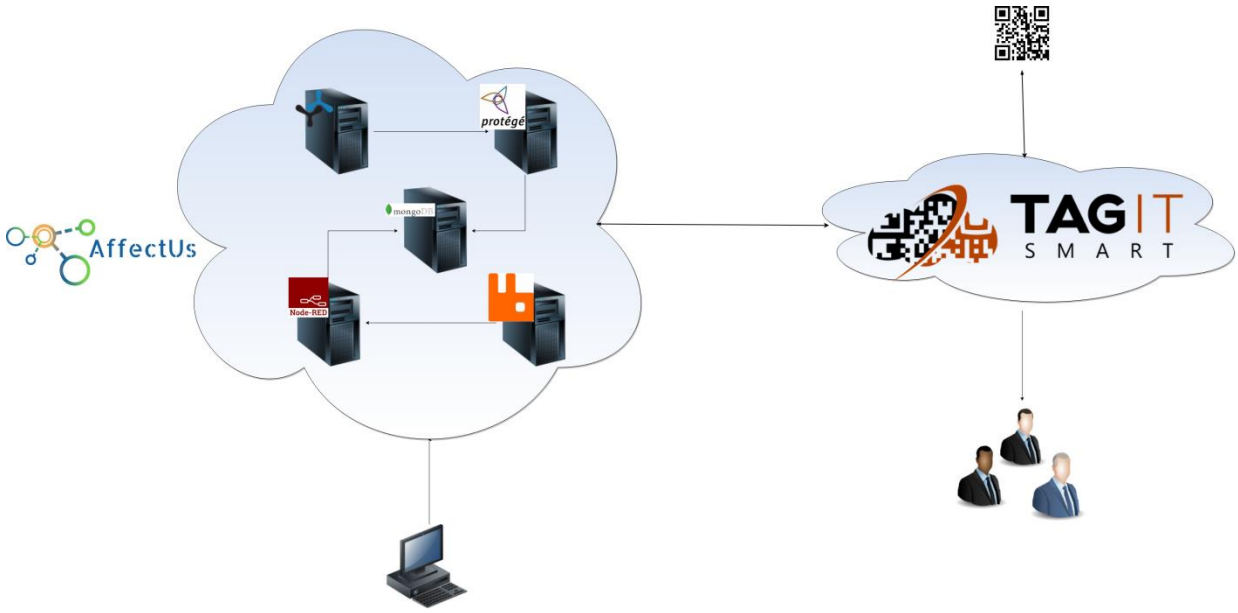


Figure 9: Positioning of AffectUs system

The AffectUs platform consists of individual software components, operating together for the common purpose. These include a database for the storing of data, events and relationships, an ontology/knowledge management system, a semantic web framework, a flow-based development tool and a message broker software tool. The development of AffectUs gives the ability to add additional features if needed.

3.2 Integrated Packaging

In order to create the AffectUs platform, free and open source software was used (Table 1). The whole platform was created using container images of the used software deployed under Docker on a swarm mode. Containers were chosen because they are lightweight, stand-alone, executable packages that contain everything needed for the software to execute (code, runtime, system tools, system libraries, settings) and isolate software from its surroundings thus reducing conflicts between different software and offering additional levels of security. Furthermore, containerization is already an industry standard giving emphasis on simplicity, robustness and portability and is designed as an embeddable component for higher level systems.

The platform was deployed for testing purposes on a Debian Linux based server (Debian 9.3 "Stretch") and the Community Edition of Docker version 17.09.1 was used. Docker was extended with the Docker compose tool that is used for defining and running multi-container Docker applications and the Docker Machine tool that allows the creation and management of nodes and swarms. A virtualized 5-node swarm was created by using the driver provided by Oracle's VirtualBox VM engine (www.virtualbox.org). The Docker images of each software where pulled directly from DockerHub (hub.docker.com) and deployed as described in the next chapter.

Table 1: List of software elements used in the AffectUs system

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

Software	Purpose	Docker Hub	Pros
Docker (Docker.com)	Containerization with independence between applications and infrastructure		• Agility • Portability • Security • Cost saving • Cloud enabled
MongoDB (Mongodb.com)	Database	Mongo (official)	• High availability • Scalability • Secure • Manageable
WebProtege (protege.stanford.edu)	ontology editor and knowledge management system	Skyplabs/webprotege	• Supports W3C standards (OWL & RDF) • Extensible • Flexible
Jena-Fuseki Server (jena.apache.org)	Semantic Web framework	Stain/jena-fuseki	• Supports SPARQL • Serves RDF data • Secure • UI for server monitoring and administration
NodeRed (Nodered.org)	Flow editor	Nodered/node-red-docker	• browser-based • Lightweight • Plethora of libraries
RabbitMQ& rabbitmq manager (Rabbitmq.com)	Open source message broker software	Rabbitmq (official)	• Reliability • Flexibility • Clustering • Federation • Highly available queues • Multi-protocol • Manageable • Traceable

The above mentioned software is the backbone of the AffectUs platform and the only software required for its operation, along with the created flows described in Section 4. For the developing and testing purposes of AffectUs, some additional software was used and deployed under the same logic of containerization (Table 2). Those include a visualizer image that shows a graphical representation of the deployed swarm, the nodes and the images that are running, a web-based management UI that allows the management and monitoring of hosts and swarm cluster, and an intermediate image that allowed to access through SSH to the Node-RED container. These images are optional, used only for developing and testing purposes, and are not required for the operation of the AffectUs platform.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

Table 2: List of external Docker images used

Image	Docker hub	Purpose
Vizualizer	dockersamples/visualizer	Visualization of the swarm, the nodes and the running containers
Portainer	portainer/portainer	Web based management user interface for standalone Docker environments and swarms
SSH for Docker	jeroenpeeters/docker-ssh	SSH Server for Docker containers, used for remotely accessing NodeRed container

3.3 System Setup Description

The AffectUs platform was build up on on a Debian Linux based server (Debian 9.3 "Stretch") in which the Docker engine, the Docker compose tool and the Docker machine tool were installed.

```
sudo apt-get install docker-ce
```

After completion, the Docker compose tool was installed

```
sudo curl -L
https://github.com/docker/compose/releases/download/1.18.0/docker-compose-
`uname -s`-`uname -m` -o /usr/local/bin/docker-compose &&\
sudochmod+x/usr/local/bin/docker-compose
```

And the Docker machine tool

```
curl -L https://github.com/docker/machine/releases/download/v0.13.0/docker-
machine-`uname -s`-`uname -m`>/tmp/docker-machine &&\
chmod+x/tmp/docker-machine &&\
sudocp/tmp/docker-machine /usr/local/bin/docker-machine
```

Then, Oracle's VirtualBox repository was added and the VirtualBox engine was installed

```
sudo apt-add-repository "deb http://download.virtualbox.org/virtualbox/debian
$(lsb_release -sc) contrib"&&\
wget-q https://www.virtualbox.org/download/oracle_vbox.asc-O- |sudo apt-key
add -&&\
sudo apt-get update &&\
sudo apt-get install virtualbox-5.1
```

After completing the above installations, we were ready to create the virtual nodes that would be used for the swarm, create a swarm itself and install all the containers as services of the swarm. Firstly, the nodes were created giving to each one of them 2GB of allocated memory. No static IP addresses were assigned since only the leaders IP should be known and all other machines would be contacted by using port forwarding.

```
docker-machine create --driver virtualbox--virtualbox-memory "2048" node1 &&\
docker-machine create --driver virtualbox--virtualbox-memory "2048" node2&&\
docker-machine create --driver virtualbox--virtualbox-memory "2048" node3&&\
docker-machine create --driver virtualbox--virtualbox-memory "2048" node4
```

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

All network addresses, including the port forwarded ones, could be hidden behind a DNS and wrapped up with an HTML frontend. This can be an item for future work and its necessity depends highly on the amount of freedom that would be given to developers and the ease of access to the backend developing tools of the platform. Firewall rules could also be used in order to block unauthorized access, but those are matters of the server infrastructure deployment and should be considered when the AffectUs platform is deployed in large scale and opened for use.

In order to create a swarm, an initialization is needed from the first machine that will be a manager of the swarm as well as the first leader of the swarm. As an output it returns the join token to be used for the nodes to join the swarm as workers. For the purposes of deploying and testing AffectUs, taking in mind that the created swarm will be very basic with only 5 nodes, all nodes joined as managers, thus after creating the first manager and advertising its IP address there was a request for creating a manager join token.

```
docker swarm init--advertise-addr<IP address of the leader>&&\
docker swarm join-token manager
```

The manager join token should be executed in every node that is wished to join the swarm. In our case the nodes were contacted through SSH

```
docker-machine ssh node1
```

and the join token was executed

```
docker swarm join --token <leader join token><leaders IP address>:2377
```

After all nodes joined, the swarm would be ready for deploying services

```
docker node ls

IDHOSTNAMESTATUSAVAILABILITYMANAGER STATUS
m3vtfmthnd7menmu0rus8fya7 *affectus   Ready   Active       Leader
w7l73ahtwynhkmlylxedn152r    node1      Ready   Active       Reachable
kuiftu9pfwapa4mqhz52d1co4    node2     Ready   Active      Reachable
rsl3etpk64hoymqar1vzz826k    node3      Ready   Active       Reachable
4ltjb0ivjfik9lk1mkb6fq90r    node4      Ready   Active       Reachable
```

At any point, even with the swarm running, a new node can join the swarm either as worker or as a manager, by simply asking the leader for a join token and executing the given command on the new machine.

Before creating the services on the swarm it is crucial to create an overlay network on top of which the services will run and communicate with each other. Maybe in a testing environment of a 5-node swarm this has not so much effect but in a larger scale it would be essential to create separate network and distinguish the groups of services that would be able to communicate. Over this logic, two networks were created and the services were joined based on their communication needs.

```
docker network create -d overlay network_A&&\
```

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

```
docker network create -d overlay network_B
```

The first service that needs to be deployed is the database instance

```
docker service create --name mongodb -d --network network_A--network
network_B--mount type=volume,source=mongodb_data,target=/data/db mongo:3
```

The data of the database are going to be stored outside the container in a folder (Volume) of the host system. This choice offers an easier way to share the volume among different containers, it is easier to backup and allows to store volumes on remote hosts or cloud providers, to encrypt the contents of volumes, or to add other functionality, as well as protect against accidental deletion of a container that would lead to data loss. Most of the services deployed afterwards will use this function for the same reasons. The database will only be used internally and there will be no ports exposed, so no setup was made to its credentials. If it is needed to access the database from a remote location then a new username and password should be declared by using the following command

```
docker exec -it mongodb mongo admin
connecting to: admin
>db.createUser({ user:'<username>',pwd:'<password>', roles:[{
role:"userAdminAnyDatabase",db:"admin"}]});
Successfully added user:{
"user":"<username>",
"roles":[
{
"role":"userAdminAnyDatabase",
"db":"admin"
}
]
```

Next, the ontology creator will be deployed, and for this purpose the webprotege was chosen as it offers an easy web-based UI.

```
docker service create --name webprotege -d --network network_A --mount
type=volume,source=webprotege_data,target=/data/webprotege -p 8888:8080
docker.io/skyplabs/webprotege
```

The webprotege service will be accompanied in the same network by the Jena-Fuseki semantic web framework

```
docker service create --name jena-fuseki -d --network network_A --mount
type=volume,source=fuseki_data,target=/fuseki -p 3030:3030--env
ADMIN_PASSWORD=affectus stain/jena-fuseki
```

The flow editor (Node-Red) will be deployed on both networks as follows

```
docker service create --name nodered -d --network network_A  --network
network_B -t -p 1880:1880--mount type=volume,source=node-red,target=/data
nodered/node-red-docker
```

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

In order to use Node-Red for the causes of AffectUs, some components need to be installed. So, after creating the container, the following commands should be run on it sequentially, after connecting to the container through SSH

```
npm install sparql-http-client
npm install isomorphic-fetch
npm install jsonschema
npm install shortid
npm install sparqljs
npm install node-red-contrib-mongodb2
npm install node-red-node-mysql
npm install node-red-dashboard
```

Then, RabbitMQ will be deployed with its manager service as a message broker service

```
docker service create --name rabbitmq -d -p 5672:5672--network network_B--
hostname affectus-rabbitmq rabbitmq:3

docker service create --name rabbitmq-man -d -p 15672:15672--network
network_B--hostname affectus-rabbitmq-man rabbitmq:3-management
```

After deploying the RabbitMQ, all services would be ready to serve the needs of AffectUs platform. For the purposes of testing and demonstration, three more services were installed, as mentioned above, and those are a visualization service, a UI container/swarm management service and an ssh service for remotely accessing the Node-RED service. These services can be ignored when deploying the AffectUs platform in larger scale.

```
docker service create --name portainer--publish 9000:9000--constraint
'node.role == manager'--network network_A--network network_B--mount
type=bind,src=//var/run/docker.sock,dst=/var/run/docker.sockportainer/portain
er-H unix:///var/run/docker.sock

docker service create --name=visualizer --publish=9090:8080/tcp--
constraint=node.role==manager --
mount=type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock
dockersamples/visualizer

docker run -d -p 2222:22-v /var/run/docker.sock:/var/run/docker.sock -e
FILTERS={\"name\":[\"^/nodered.1.7xyta4naeegyxcskfyf88ilvq$\"]} -e
AUTH_MECHANISM=simpleAuth -e AUTH_USER=root -e
AUTH_PASSWORD=affectusjeroenpeeters/docker-ssh
```

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

After completing the deployment, the services can be accessed by using the leaders IP address and the corresponding port:

Node-Red: <Leaders IP Address>:1880
Node-Red SSH access: <Leaders IP Address>:2222
Web-Protege:<Leaders IP Address>:8888
Jena FusekiServer:<Leaders IP Address>:3030
RabbitMQ Daemon: <Leaders IP Address>:5672
RabbitMQ Manager: <Leaders IP Address>:15672

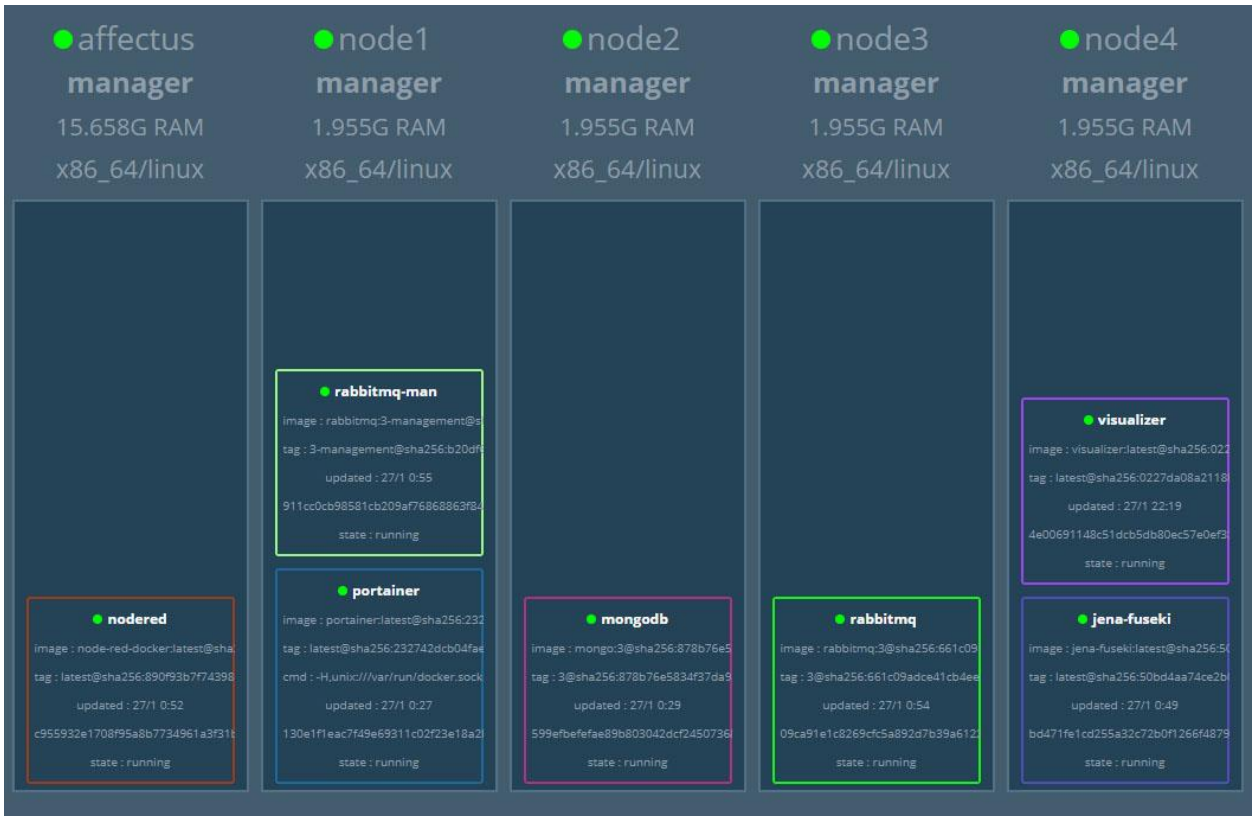


Figure 10: A graphical representation of the running services, given by the visualization service

3.3.1 Single Step Deployment Process

All of the above steps that were described were included primarily as a generic guide for audiences to understand the process. ***However the overall deployment of AffectUs can be concentrated and grouped to a single compose (YAML) file for easier deployment, better scalability, preservation of containers data, easy recreation and update of the existing containers and more efficient management***. The following compose file contains all the aforementioned services and those that are not needed for production (portainer, visualize, ssh for docker) could be removed before executing.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

```
version: "3.1"
services:

mongodb:
    image: mongo:3
    volumes:
      - mongodb_data:/data/db
    networks:
      - network_A
      - network_B
    deploy:
      replicas: 1
      update config:
        parallelism: 2
        delay: 10s
restart_policy:
        condition: on-failure

webprotege:
    image: docker.io/skyplabs/webprotege
    volumes:
      - webprotege_data:/data/webprotege
    networks:
      - network_A
    ports:
      - 8888:8080
    links:
      - mongodb
    deploy:
      replicas: 1
restart_policy:
        condition: on-failure

jena-fuseki:
    image: stain/jena-fuseki
    volumes:
      - fuseki_data:/fuseki
    networks:
      - network_A
    ports:
      - 3030:3030
    environment:
      - ADMIN_PASSWORD
    deploy:
      replicas: 1
restart_policy:
        condition: on-failure

nodered:
    image: nodered/node-red-docker
    volumes:
      - node-red:/data
    networks:
      - network_A
      - network_B
    ports:
```

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

```
      - 1880:1880
    links:
      - mongodb
    deploy:
      replicas: 1
restart_policy:
        condition: on-failure

rabbitmq:
    image: rabbitmq:3
    networks:
      - network B
    ports:
      - 5672:5672
    hostname: affectus-rabbitmq
    deploy:
      replicas: 1
restart_policy:
        condition: on-failure

rabbitmq-man:
    image: rabbitmq:3-management
    networks:
      - network_B
    ports:
      - 15672:15672
    hostname: affectus-rabbitmq
    links:
      - rabbitmq
    deploy:
      replicas: 1
restart_policy:
        condition: on-failure

portainer:
    image: portainer/portainer
    networks:
      - network_A
      - network_B
    ports:
      - 9000:9000
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    deploy:
      placement:
        constraints: [node.role == manager]
      replicas: 1
restart_policy:
        condition: on-failure

  visualizer:
    image: dockersamples/visualizer
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    deploy:
      placement:
        constraints: [node.role == manager]
```

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

```
    ports:
      - 9090:8080
    deploy:
      replicas: 1
restart_policy:
        condition: on-failure

nodered_ssh:
    image: jeroenpeeters/docker-ssh
    networks:
      - network_B
    ports:
      - 2222:22
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    environment:
      - FILTERS
      - AUTH_MECHANISM=simpleAuth
      - AUTH_USER=root
      - AUTH_PASSWORD
onrun:
      - npm install sparql-http-client
      - npm install isomorphic-fetch
      - npm install jsonschema
      - npm install shortid
      - npm install sparqljs
      - npm install node-red-contrib-mongodb2
      - npm install node-red-node-mysql
      - npm install node-red-dashboard
deploy:
      replicas: 1
restart_policy:
        condition: on-failure

networks:
network_A:
network_B:

volumes:
mongodb_data:
webprotege_data:
fuseki_data:
  node-red:
```

Figure 11: Single step deployment descriptor (Docker Service YAML file) for the AffectUs system

With the compose file available, the creation of the services could be done with a single command as follows (assuming that the swarm is ready with all nodes available)

```
eval"$(docker-machine env --swarm <name of swarm leader machine>)"&&\
docker-compose -f <name of file>.ymlup
```

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

3.4 License and Contact

All software components of AffectUs are released as open source code, following the Apache V2 licensing model. Code artefacts will be included on Github while the Docker images will be included in public repositories, following their finalization at the end of the IoT challenge/hackathon to be organized in the final month of the project.

For questions or help during the usage of the tools, send emails at affectus(at)hua.gr.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

4 Component Description and Manual

4.1 UI Elements

The UI elements of AffectUs undertake the role of simplifying the interactions of the envisioned roles as described in D1.1 (Product Stakeholder, Application Developer, Analytics Developer) with the AffectUs extension, in order to declare and initialize a minimal set of required information from their side. These UIs are described in detail in the following subsections along with their functionality and relation to the defined use cases of D1.1.

4.1.1 General UI Setup

The top level setup is depicted in Figure 12, indicating the available pages. Emphasis has been given in the role separation of the UI, so that it is more user-friendly and relevant to each entity. From this main screen a user can redirect to any of the supported pages based on their intentions.

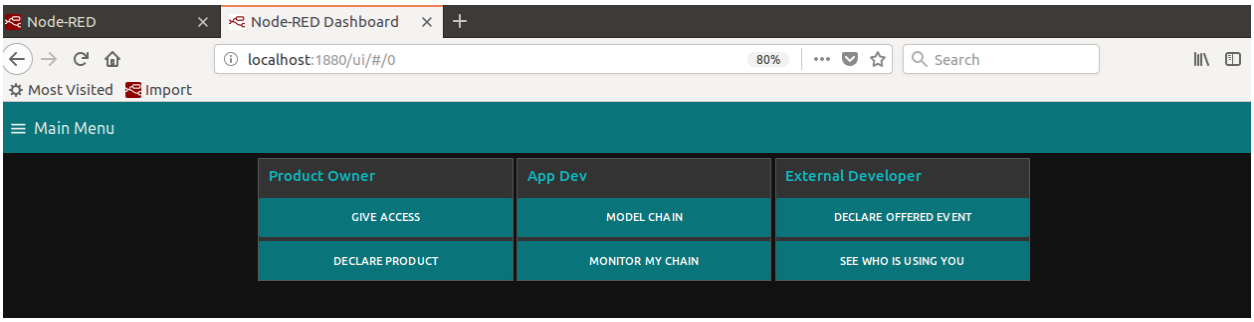


Figure 12: Top Level Main Menu Setup

In order to aid them in the navigation, a side panel is also available with all the relevant options, again highlighting the roles per case.

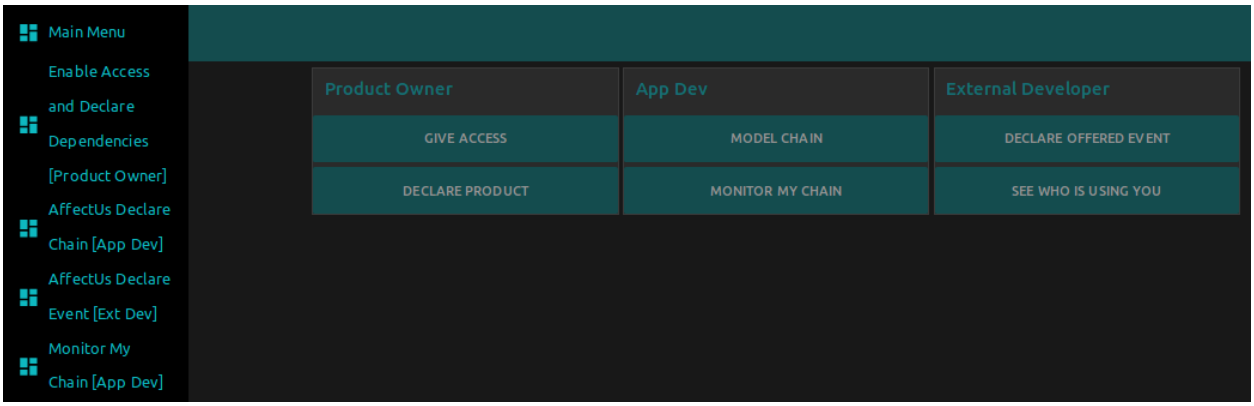


Figure 13: Side Navigation Panel of UI

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

4.1.2 Product StakeholderUI (Enable Access and Declare Dependencies)

4.1.2.1 Supported functionalities and Usage

With relation to the original figure from D1.1 (Figure 14 below), the intended covered functionality appears in the red highlighted area and has indeed been extended to cover for the case of Application Key declaration by the Product Owner towards the AffectUs extension, in order to enable OATH based access.

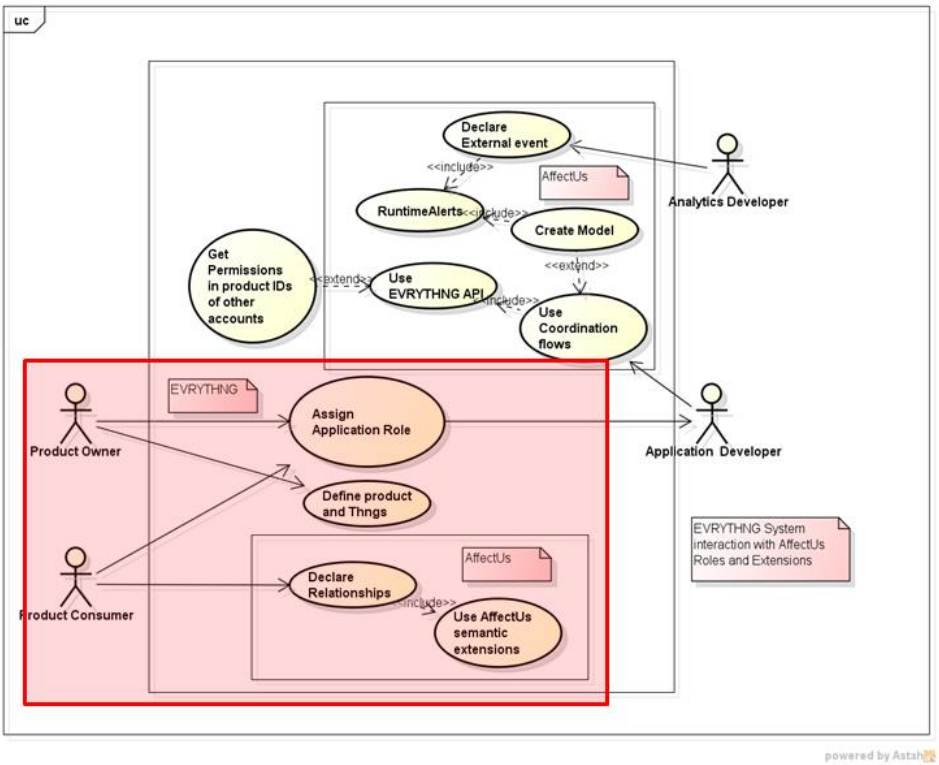


Figure 14: Generic Role interaction with the AffectUs system (Product Stakeholder UI coverage)

In detail, the product stakeholder, product owner or (business) consumer, initially needs to perform the following operations

- Enable reduced, OATH based access of AffectUs to their EVRYTHNG account in order for our extension to be able to retrieve their product list and also get real time data regarding these
- Declare dependencies of their products from both external conditions and domains (declared through this UI after retrieving the respective categorizations from the KB) or other products (done indirectly through the definition of the supply chain structure in Section 4.1.3 by the supporting Application Developer). This way we will be able to correlate during runtime events that occur in one of the dependencies and propagate it to the affected entities.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

- Define notification endpoints in which the respective information will be published

These are all included in the main Product Owner UI (Enable Access and Declare Dependencies), which appears in Figure 15.

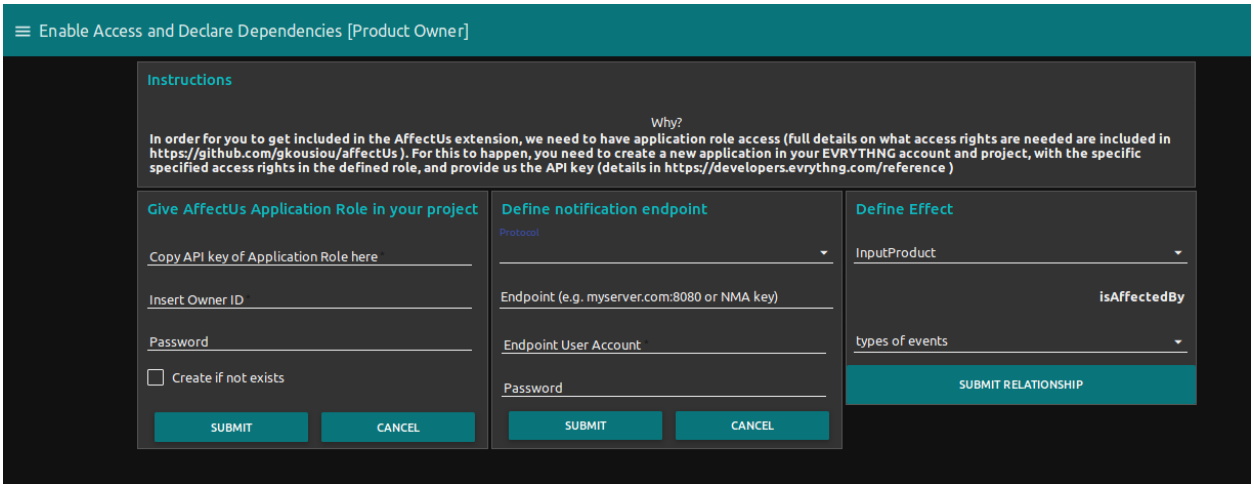


Figure 15: Product Owner Enable Access UI

Give AffectUs Application Role in your project Tab

In this panel, the Product Owner needs to copy the Application Key that they create through the EVRYTHNG platform (more details on https://developers.evrythng.com/reference). During this operation they can also define the access rights that AffectUs will have on their account, in terms of acquired data and granularity. The minimum needed rights are included in Section 4.1.2.2.2. The Owner ID and password are NOT the ones on EVRYTHNG, but on the AffectUs platform. We do not require any specific access to the owner's EVRYTHNG account other than the OATH based one.

Define Notification Endpoint

In this tab the Product Owner needs to declare the endpoint to which they want to receive notifications regarding their product, either coming from their own supply chain or from external events that might affect them. Necessary input includes selection of one of the following protocols

- HTTP POST service endpoint
- AMQP endpoint
- MQTT endpoint
- NMA endpoint, for which the NMA key is needed in order to forward the message

For the cases that require authentication (except for NMA), the necessary credentials need to be provided. The acquired information is stored in the KB for future use.

Define Effect

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

In this tab the Product Owner needs to define the domains of events from which they would like to be informed about. Therefore, if for example their product is affected by weather, this should be declared in this tab, while the AffectUs backend will correlate events that are nearby (in terms of space and time) their products coming from this weather domain. This is the same categorization also used by the External Analytics developer in order for correlation to be feasible. The enumeration of the product options is populated automatically by AffectUs once the first step of Access grants is performed in the "Give AffectUs Application Role in your project" tab**.** The domain options are retrieved from the KB. The overall goal of this tab is to enable the Product Owner to concretize relations of the sort "productAisAffectedBy weather", necessary to be included in the backend KB and according to the structure needed by the defined ontology.

4.1.2.2 Implementation Details

The UIs as well as the functionalities are implemented as Node-RED flows. More information is provided in the following sections.

4.1.2.2.1 Node-RED implementation and Integration with AffectUs Backend

The complete Node-RED flow implementation appears in Figure 16. Apart from some information that is populated statically upon initialization (such as the list of supported protocol endpoints), the remaining flow is fully dynamic. As mentioned in the previous section, initially the Product Owner needs to copy an application key (created within the EVRYTHNG environment) in the UI and submit. Upon this action the flow contacts through an HTTP call the EVRYTHNG platform and retrieves the product list associated with this key (red box in Figure 16 indicated as "integration with TIS").

Furthermore, this information needs to be persisted in the AffectUs backend (MongoDB instance), in order to store the mapping between products and relevant app key to maintain state needed for further calls (indicated by the blue annotation box "Integration with AffectUs backend-MongoDB").

Following, and based on the retrieved information, the triples need to be created that will be stored in the AffectUs backend KB. From this UI the following triples are created:

- Producer produces product (created from each element of the retrieved product list of the product owner)
- Actor hasEndpoint [input_endpoint] (from the declared endpoint)
- Actor hasCredentials [user,pwd] from the declared ones
- Product isAffectedBy EventType

Context flow variables are used in each case when a triple needs to be prepared in multiple steps with inputs from various submission forms.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

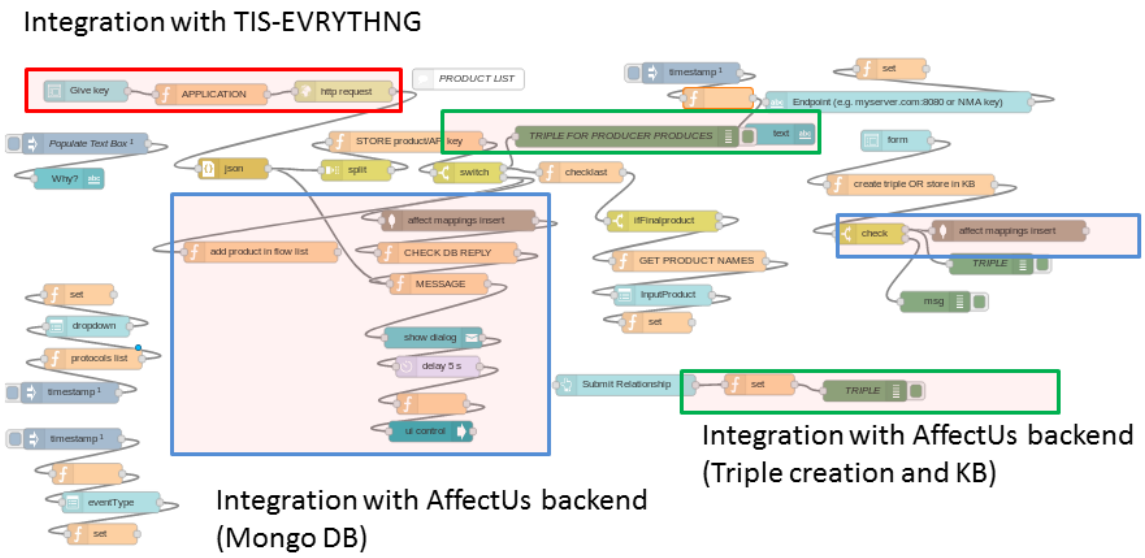


Figure 16: Node-RED implementation of Enable Access UI

4.1.2.2.2 Integration with TIS- Enablement of Application Role for AffectUs in Product Owner's EVRYTHNG account

In order for the products to be available for inclusion in the AffectUs ecosystem, the respective Operator (EVRYTHNG role) of the EVRYTHNG account of the specific products needs to perform the following steps:

- Create a project inside the EVRYTHNG platform
- Create the product class that will be shareable with AffectUs inside that project
- Create the AffectUs application role with specific permissions (mainly read-only) and the respective application through the EVRYTHNG dashboard. The base application role template is sufficient[5], if one needs even less rights to be given to external entities we can remove all create and delete rights in the various OATH resources.
    - During this process an application API key is created and portrayed in the dashboard
- Go to "Account Operator Enable Access" tab of AffectUs front end, copy the application API key and submit the information
- Upon submission, the AffectUs backend retrieves the product or other resource list that will be used across the AffectUs lifecycle and maintains the mapping between these resources and the specific API key for future usage in a backend MongoDB instance. Thus upon need to perform

any operation relevant to that resource, the respective key is retrieved and added as a parameter in the request URL.

For more information on the first 3 steps visit the respective EVRYTHNG documentation.

4.1.2.2.3 Information templates

Information templates needed for this case include two aspects:

- The KB format that needs to be followed and is detailed in Section 4.2
- The format of the products coming from EVRYTHNG and appears in


```
▼ 0:
    id:             "UHdxGdtgegsatpwRwhD7ghyr"
    createdAt:      1513198582991
    customFields:   {}
    updatedAt:      1513198582991
    brand:          "siemens"
  ▼ properties:
      second:       55
    description:    "second prod class"
    fn:             "myprod2"
    name:           "myprod2"
    identifiers:    {}
▼ 1:
    id:             "UnUbC8B7BXPwtpawRhGspr4d"
    createdAt:      1513184845111
    customFields:   {}
    updatedAt:      1513184845111
  ▼ properties:
      dfdfdfd:      5
      sds:          88
      sssssssss:    22
      value:        25
    fn:             "myprod1"
    name:           "myprod1"
    identifiers:    {}
```


Figure 17: Product format coming from TIS EVRYTHNG

4.1.2.2.4 Other Helper flows

N/A

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

4.1.3 Product Stakeholder UI (Chain History)

4.1.3.1 Supported functionalities and Usage

The chain history interface is visualizing historical information about a specified production chain, associated with a product. This enables a product stakeholder to identify strong and weak points in the chain, taking proper actions on both cases. The stakeholder can choose any production chain and any time period, having the interface create charts for each stage of the chain in real time. Any change in the time period or the product will be applied, also in real time, changing the charts with the relevant data, retrieved from the historical database. More specifically, the data are graphically presenting the number of products being in each stage, distinguishing between the products that have stayed for too long in this stage and the products that are moving normally through it.

In the general use case, described in D1.1, this interface would implement one aspect of the EVRYTHNG API usage that serves important statistical data to a product Stakeholder, in visual format. This relation to the general use case is shown in the following figure:

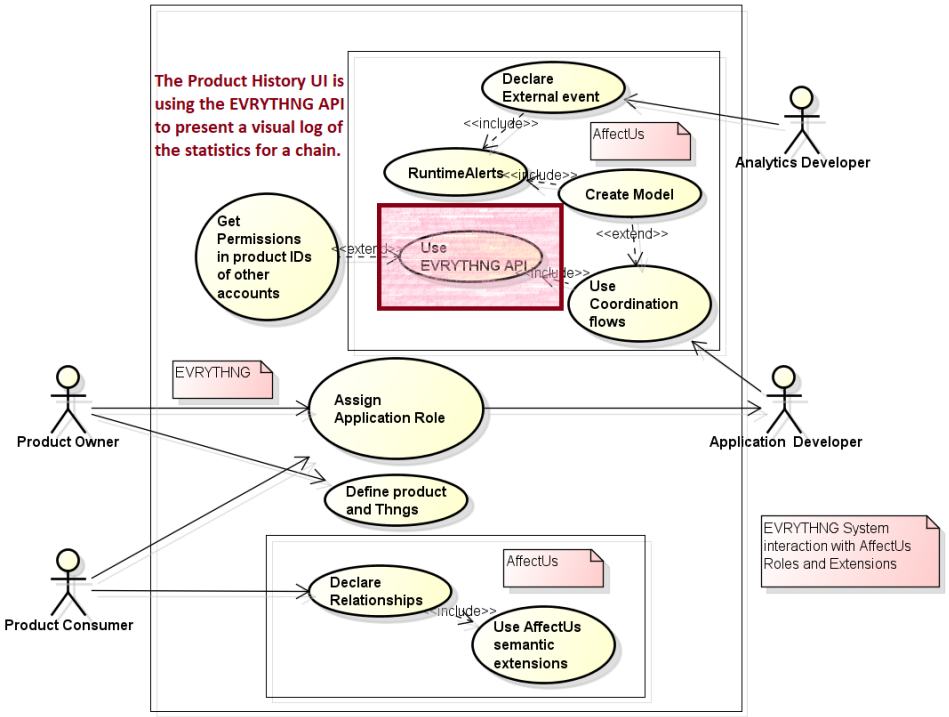


Figure 18: Chain History UI in the general use case

4.1.3.2 Implementation Details

4.1.3.2.1 Node-RED implementation and Integration with AffectUs Backend

The Chain History UI is fully implemented in Node-RED, receiving three input parameters, the product ID which associates the monitored data with a specific product chain and the starting and end date setting the time period to monitor. The connection to the AffectUs Backend and the other components of the

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

system is made through the MongoDB. Every time a product is either changing stage or surpassing the expected duration in a stage, a new event is raised, updating the associated counters in the database. Then, the interface is requesting the relevant data from the database every time one or more of the inputs is changed.

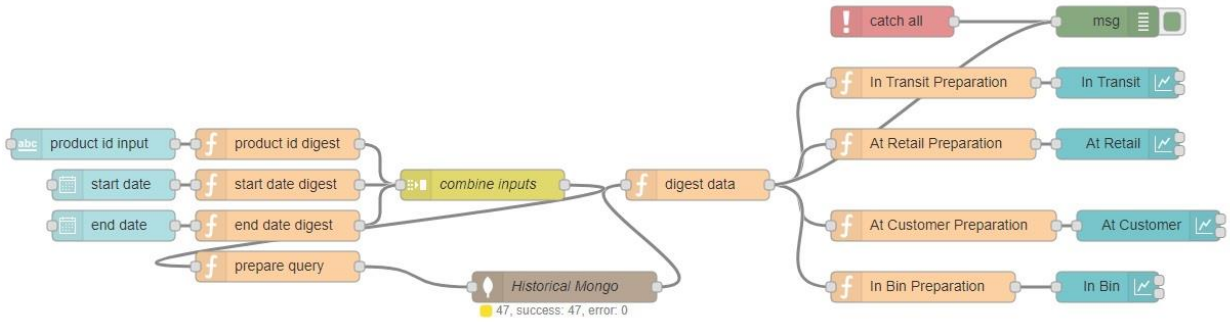


Figure 19: Chain History Node-RED flow.

All the data are gathered and presented in chart format, split per chain stage. That enables the product stakeholder to fiddle with the input parameters and explore her production chain for weaknesses and anomalies that need to be addressed. It also enables her to identify strong points in the chain that need to be preserved.

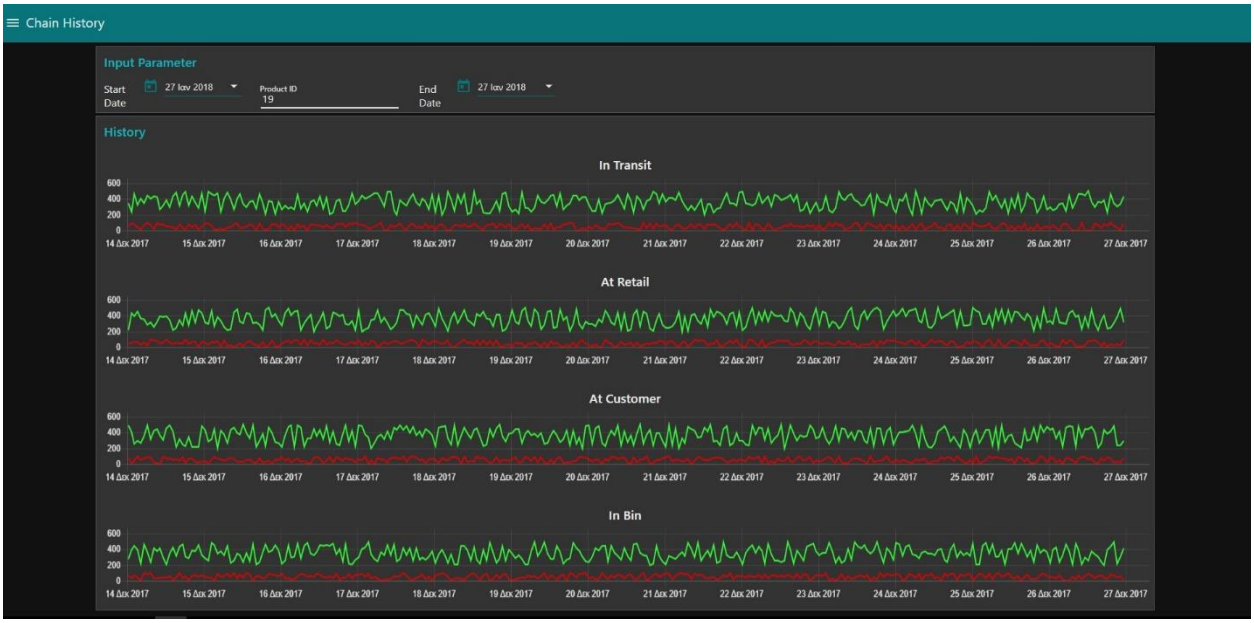


Figure 20: Chain History User Interface.

4.1.3.2.2 Integration with TIS

The chain history UI is not directly integrated with TIS but it uses data derived from its functionality. Specifically, it uses the data mentioned earlier, containing information about the number of products that are in each stage of the production chain and the time they spend in it. These data are made available through the product scans, using the TIS technology.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

4.1.3.2.3 Information templates

The interface by itself is not producing any information but it is using data drawn by the MongoDB in the following JSON format:

```
{
"timestamp":1514323236073,
"data":[
        {"stage":"in_transit","products_normal":436,"products_delayed":82},
        {"stage":"at_retail","products_normal":256,"products_delayed":5},
        {"stage":"at_consumer","products_normal":453,"products_delayed":69},
        {"stage":"in_bin","products_normal":468,"products_delayed":78}]
}
```

Figure 21: JSON example of retrieved historical chain status from MongoDB

4.1.3.2.4 Other Helper flows

In order to store the necessary data, the interface is employing a helper flow, monitoring the timeout and stage change events and updating the relevant counters in the database.

4.1.4 App Developer UI (Declare Chain)

4.1.4.1 Supported functionalities and Usage

With relation to the high level generic roles interaction with AffectUs from D1.1, appearing in annotated form in Figure 22, this section aims at describing the highlighted red rectangle, consisting of the main interactions of the Application Developer in order to create a model of the product owner's supply chain.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

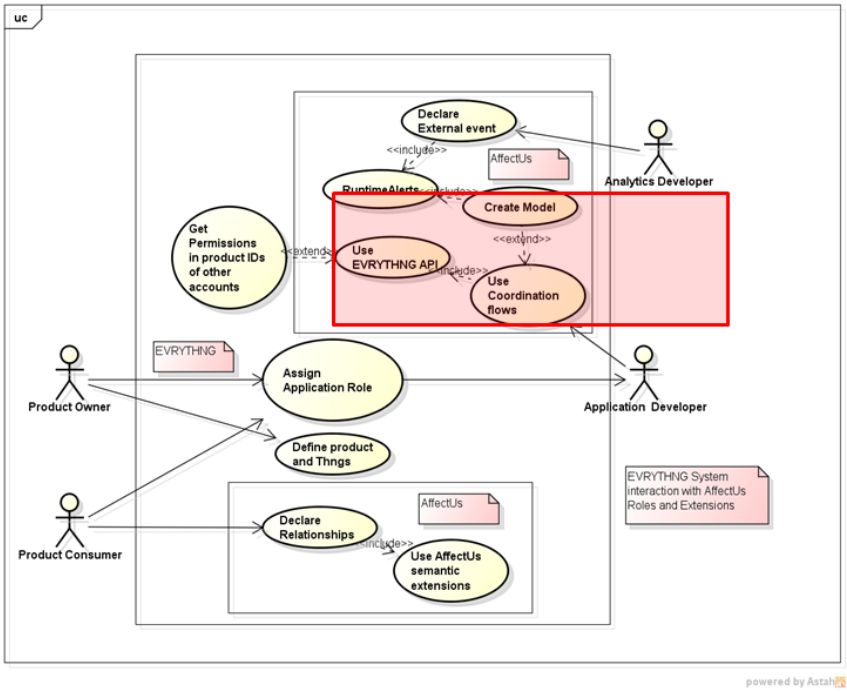


Figure 22: Generic Role interaction with the AffectUs system (App Developer UI coverage)

The main UI implementing the declaration of the supply chain stages appears in Figure 23. Initially the App Developer declares the chain name, which is the main identifier, with which the individual chain stages and other details are linked.

Define Supply Chain Stage Options tab

Following the "Define Supply Chain Stage Options tab" is populated. The type is selected from a generic and static enumeration of stage types ("out of production", "In transit", "At Retail", "At End user", "At recycle bin" etc.). This enables the creation of the supply chain models afterwards. For each stage the input and output products are declared, from a retrieved list from the KB. Products may refer to the same product owner or to another owner (indicating that these act as raw materials in this chain). A product may be the same in the input and output, if for example this is in transit stage.

Define Supply Chain Stage Inputs tab

After the definition of the stage type and involved products, the App Developer needs to model the specific inputs necessary to identify this stage, such as the stage name as well as location information (as GEOJSON objects). This will enable us to retrieve incoming scan data, get the GPS coordinates and categorize the scan to one of the stage types. GPS sensitivity is also defined, in the sense that the location may be a center and the sensitivity the defined radius around the point that indicates the stage's area.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

One critical aspect is the declaration of the previous and next stage's names (as given in their respective declarations), given that based on this following events may be detected. Examples of these have to do with identifying illegal states of a product, creation of the model chain etc.

Finally the submission tab indicates the created triples and the submission status.

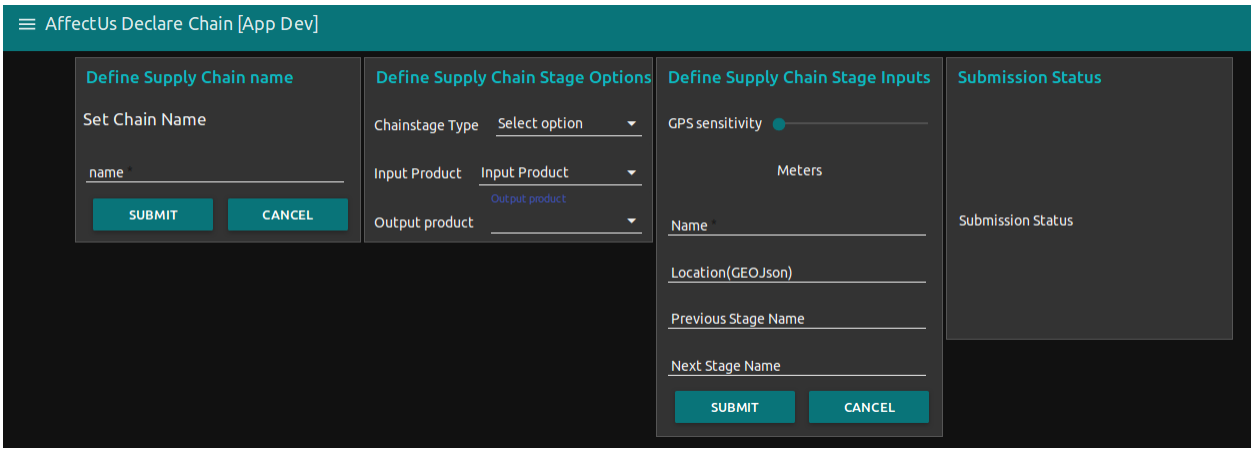


Figure 23: App Developer Declare Chain UI

4.1.4.2 Implementation Details

4.1.4.2.1 Node-RED implementation and Integration with AffectUs Backend

The flow for implementing the aforementioned functionalities and link with the backend appears in Figure 24. Initially through the Chain name declaration and submission the relevant triple is created and stored directly (Set Chain Name node and following flow). This results in the creation of the following triple:

[InputChainName] rdf:type SupplyChain

Then the App Developer declares each stage individually. Upon population of the input,output products and selection of chainstage type (from an enumerated list of potential stage types), flow context variables are initialized ("set " nodes). The same applies for the location sensitivity bar. Location GEOJSOn is copied in the respective text box. Upon stage submission the following triples are created and submitted to the KB:

[input.name] isA chainstage *[input.name] isOfType [input.stageType]* *[input.name] belongsTo flow.get("supplyChain")* *[input.name] hasInputs [input.inputProduct]* *[input.name] hasOutputs flow.get("outputProduct")*

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

[input.name] hasLocation [input.locationGEOJSON] *[input.name] hasPrevious [input.previousStageName]* *[input.name] hasNext [input.nextStageName]*

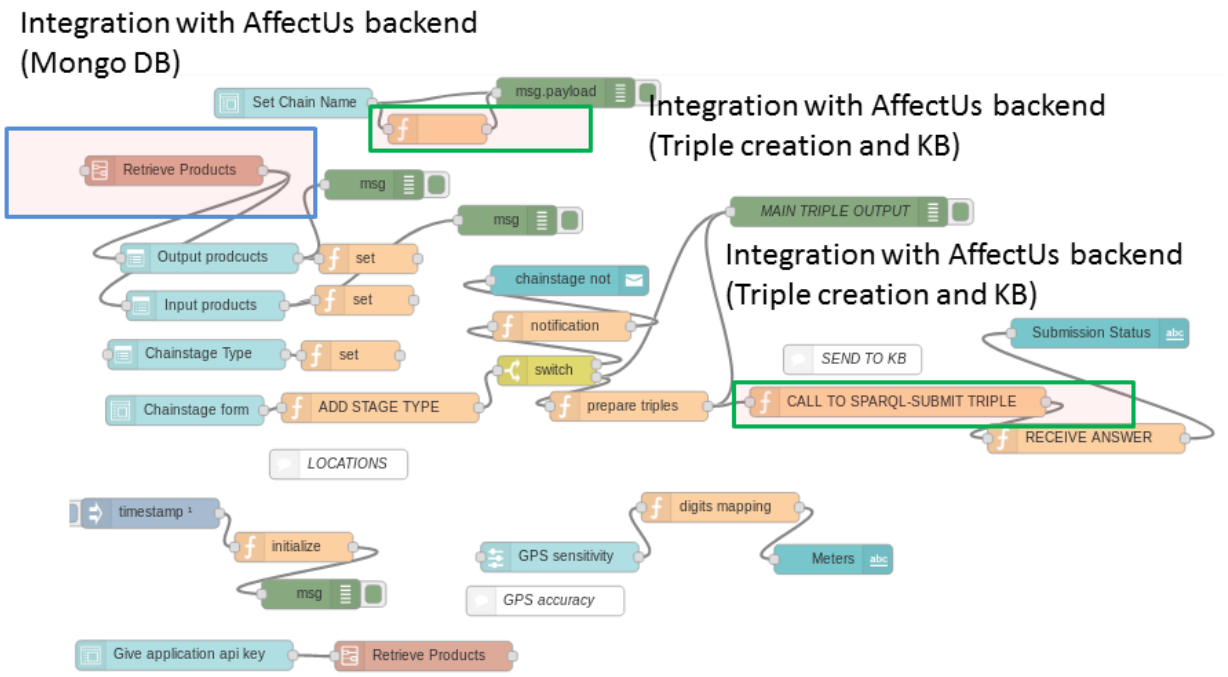


Figure 24: Node-RED flow for implanting the App Dev Declare Chain functionality

4.1.4.2.2 Integration with TIS

In the specific case there is no integration with TIS since TIS related data (e.g. product lists) have already been included in the AffectUs MongoDB backend. Furthermore, mapping from the TIS scan data is transformed to the chainstage type through the respective linking flow (Section 4.4.1).

4.1.4.2.3 Information templates

Information templates related to this case include:

- The enumeration of the chainstage types:
  - Out of production
  - In Transit
  - At retail
  - At end user
  - At recycle bin
- The format of the location definition of the chainstage. For this case the GEOJSON format is assumed[4].

4.1.5 App Developer UI (Monitor My Chain)

4.1.5.1 Supported functionalities and Usage

This interface is visualizing, in real time, the status of each stage in a chosen production chain. This status contains the amount of products that are currently in each stage, clearly distinguished as delayed or normal. The delayed products are those that have surpassed a timeout threshold in their current stage whereas the normal ones are those that are normally crossing the stage. This helps the app developer to monitor the chain, locate errors in it that are delaying products from moving to the next stage and visualize a live feed of information, possibly helping the developer debug an application.

The interface fits in the general use case diagram, presented in D1.1, in the coordination flows box, by using the model output and the EVRYTHNG API in order to assist developers in their job. This is shown in the following figure:

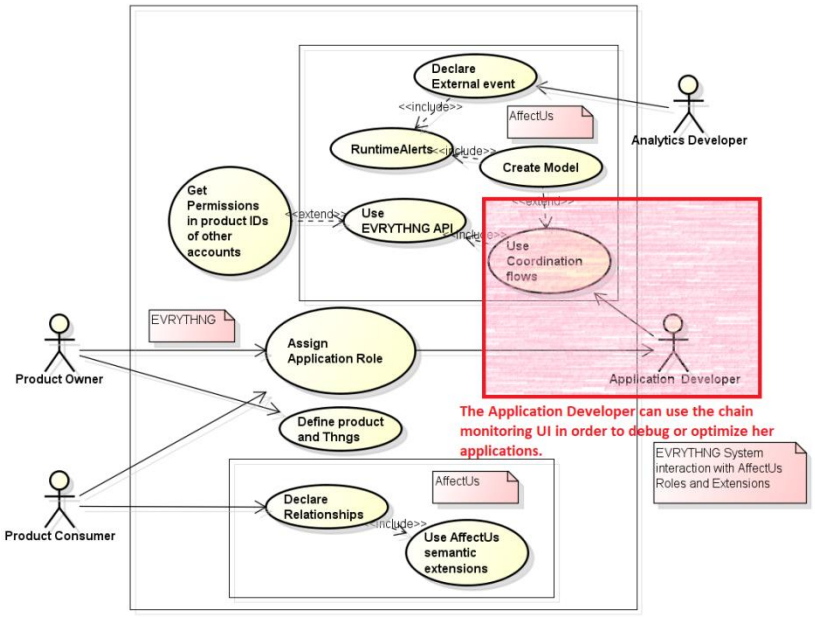


Figure 25: Chain monitor UI in the general use case.

4.1.5.2 Implementation Details

4.1.5.2.1 Node-RED implementation and Integration with AffectUs Backend

The interface is fully implemented in Node-RED, using the dashboard node to process a live feed of data and visualize it in a browser environment. The flow is taking in a periodically updated stream as input, processes the data, limits them to only the product ID chosen by the user and then presents them in a graphical user interface to the user, providing advanced insight in the real time status of the production chain. The interaction with the AffectUs backend is done through a web socket, by having a helper flow monitor for any stage change or product delayed event and sent it through the web socket as a periodically updated stream.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

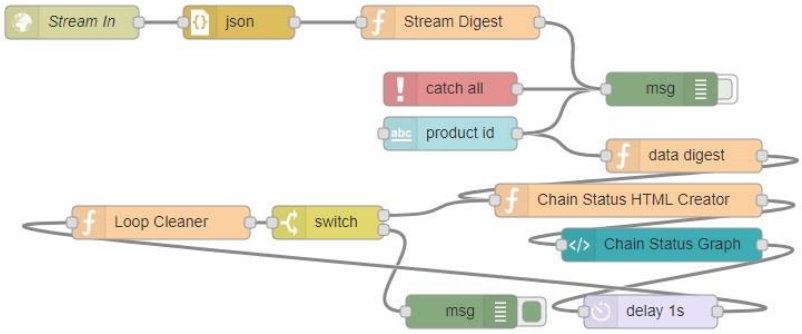


Figure 26: Chain Monitor Node-RED flow

The User Interface is graphically presenting the current status of the production chain by updating the number of products currently in each stage of the chain. As shown in the following figure, the number of products in each stage is separated in products normally crossing the stage (green) and the products that have surpassed the expected duration of remaining in this stage (red). The normal products present to the user an estimation of the production levels whereas the delayed products present possible bottlenecks or inefficiencies in the chain.

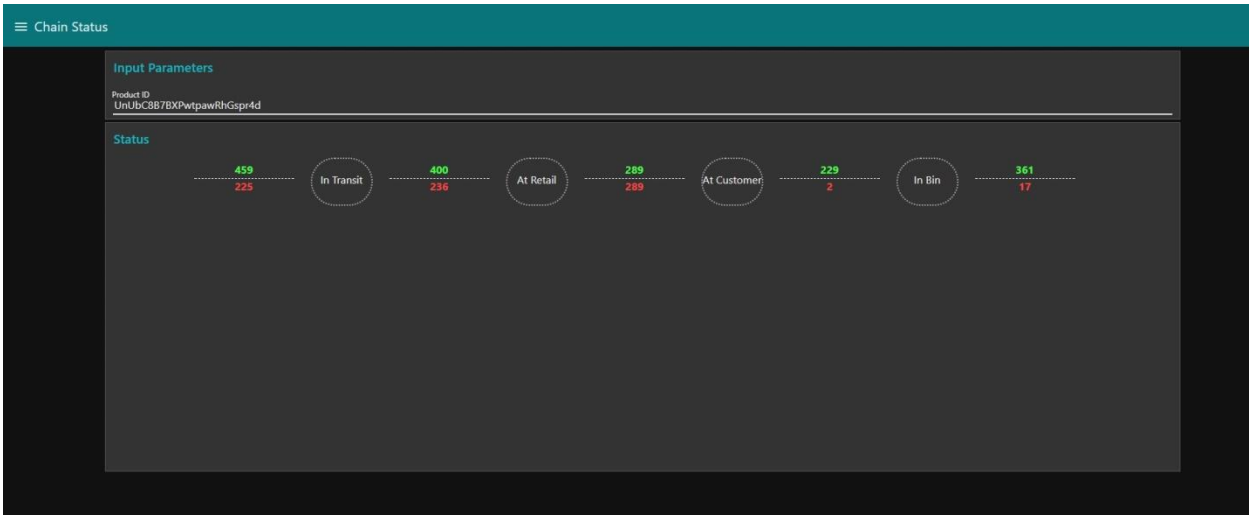


Figure 27: Chain Monitor Interface

4.1.5.2.2 Integration with TIS

The integration with TIS is done through the helper flow that monitors for stage change events. If a new scan arrives, the helper flow updates the relevant thing's stage and creates a stage change event, whereas if the expected time duration passes by without having received the appropriate stage change event, a delay event is created, notifying the platform that the thing is spending more time than expected in its current stage.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

4.1.5.2.3 Information templates

The stream input expected by the flow is a JSON array containing counters for each of the tracked products. Each element of the array contains all the information relevant to a specific product ID, including the number of things in each stage of the chain and their status (either normal or delayed). An example of the template is presented here:


```
[{
"in_transit":[
        {"thing_id":c4ca4238a0b923820dcc509a,"timestamp":1517315733615,"delayed":false},
        {"thing_id":1c9ac0159c94d8d0cbedc973,"timestamp":1517316238894,"delayed":false},
        {"thing_id":b76d3125a661028425ecf5de,"timestamp":1517316620626,"delayed":true}
],
"at_retail":[
        {"thing_id":f3bdbacdbeb54aad330fbc7e,"timestamp":1517315808958,"delayed":false},
        {"thing_id":912f6e1b2cdeba1de3176bcf,"timestamp":1517316587480,"delayed":true},
                {"thing_id":c5a8fd4ee41b2c4efe718839,"timestamp":1517316839601,"delayed":false}
],
"at_customer":[
        {"thing_id":e385e9c59ac5151a5bf6fcbd,"timestamp":1517316390572,"delayed":false},
       {"thing_id":b1b7454db08d98ed5f52479,"timestamp":1517316501084,"delayed":false},
        {"thing_id":a2a375a8ec6921943bbb811,"timestamp":1517316527203,"delayed":false},
        {"thing_id":626a7d175e2947bd13b321c,"timestamp":1517316586474,"delayed":false},
        {"thing_id":88ccee08deefb5ff5073b757,"timestamp":1517317168076,"delayed":false},
        {"thing_id":67832d2b34a540655f4319e1,"timestamp":1517317380048,"delayed":false}
],
"in_bin":[]
}]
```


Figure 28: JSON example of Chain Monitor

4.1.5.2.4 Other Helper flows

The chain monitor is supported by a helper flow that monitors the AffectUs platform for stage change and product delayed events. These events trigger an update in the stored counters and an update is sent through the stream web socket in order to be processed by the chain monitor user interface.

4.1.6 External Analytics Developers UI (Declare Provided Events)

4.1.6.1 Supported functionalities and Usage

The user interface visualizes the options that the external analytics developer has available. These options enable the developer to essentially create the process of their event detection in whatever manner they see fit. Through the “Declare Event Type” of the UI, they declare the type of event detection they want, by selecting the type of external events they want to incorporate, by declaring the name for the condition in which an event occurs, the external source used for detecting the event. To force analytics developers to give meaningful data we give options for including timestamp and location,

and impose that they provide at least one of the options. We also include the declaration of measurement units, and an optional URI that points to either an alternate description or application specific details about the event that the developer may wish to include.

Through the “Declare Endpoint” section the external analytics developer can declare the protocol and the endpoint with which the events are published. They also provide the JSON description for the event publication. The JSON is validated in real-time and a notification message is returned regarding errors missing fields and overall validity. All the inserted data is deduced to subject-predicate-object triples that can be checked for their validity in the “Check Triples” section. Finally when everything is in order, they can submit the triples. This way, the workflow for the external developer is streamlined.

4.1.6.2 Implementation Details

4.1.6.2.1 Node-RED implementation and Integration with AffectUs Backend

The interface is fully implemented in Node-RED, using the dashboard node to provide the options and the fields available to the developer. The flow checks the given input in real time in JSON Description and upon request in the “Check Triples” section. The UI context is used as temporary storage for the inserted data. To check all the inserted structures in the entirety of the flow we use JSON schema.

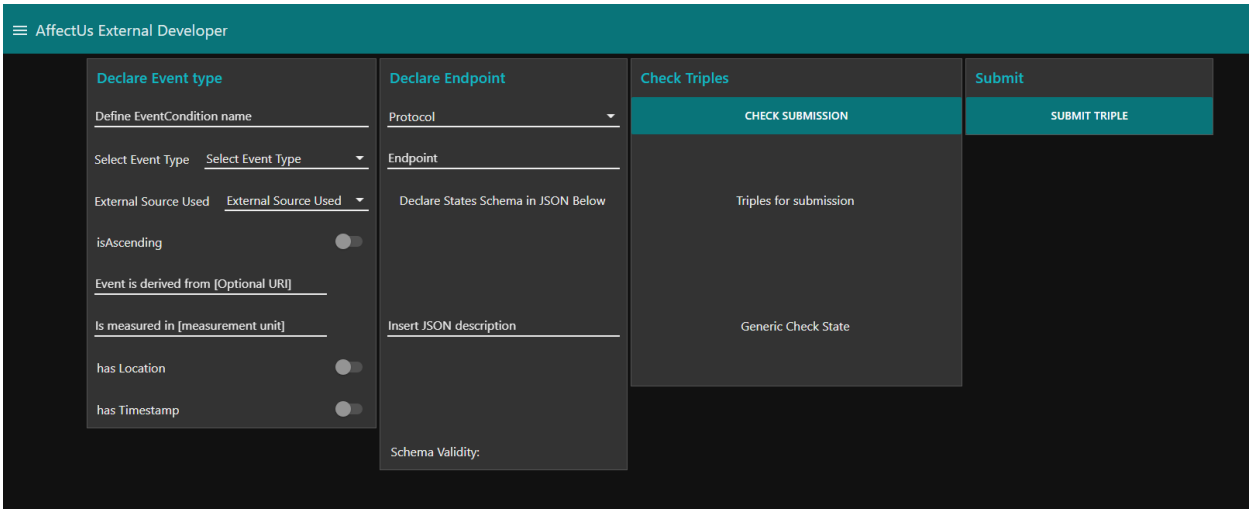


Figure 29: External Analytics Developer UI

If the check is performed and the “Check State” is order then knowledge extracted from this UI is then stored in the KB, so that other flows can use detected events or parameters associated with the process of event detection. When the submission is successful and the process is complete, a message is returned, and a new set of triples can be added by the developer. The flow is shown below.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

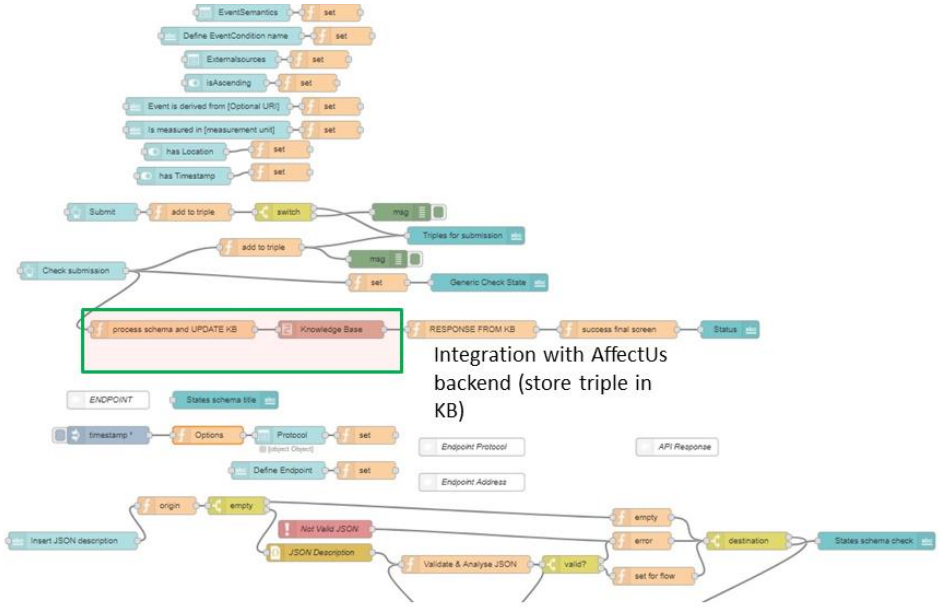


Figure 30: External Analytics Developer Flow

4.1.6.2.2 Integration with TIS

The integration with TIS is not direct but through the creation of message structures for events, and declaring the sources of events for helper flows. This UI also provides the parameters with which the events or messages are sent. We accomplish this by populating the KB with the required knowledge for event publication and extraction, used by other helper flows and by the application developer and product stakeholder UIs.

4.1.6.2.3 Information templates

For the JSON that the analytics developer provides in the UI, there is a Schema that is followed. The JSON expected in that input, so that it is valid is an array of objects. There has to be at least one object in the array. Each object is essentially a state for the event condition. Inside each state there exist the name and the cluster parameters as an array of objects. The array consists of cluster parameters. The cluster parameters are a name for the feature of the state, and minimum, maximum and optionally average numeric values for that state. As such the JSON input follows this template:

```
[
        {"stateName" : "Cold",
                "stateParams":[
                        {"clusterParameterName":"Temperature",
                        "min":15,
                        "max":30
```

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

```
                    "average":20
                    }
                ]
        },
        {"stateName" : "Warm",
        "stateParams":[
                    {"clusterParameterName":"Temperature",
                    "min":15,
                    "max":30
                    "average":20
                    }
                ]
        }
]
```

Figure 31: JSON example of States Declaration

There can be many more than cluster parameters, as a state may consist of many, e.g. both Temperature and Humidity.

4.2 Semantic framework and service

4.2.1 Finalized semantic structure and OWL

4.2.1.1 Ideas behind Semantics and Usage Overview

AffectUs relies on the interconnection of complex concepts. For that reason we specify a vocabulary that suits our needs, and we connect them so that they can produce new meaning, both conceptually and programmatically. This is why we use some of the most mature Semantic Web technologies to be the basis of our knowledge base. For the semantic structure we use the Web Ontology Language (OWL). OWL suits our need because it is based on Description Logics knowledge representation formalism. Due to the complex structure required, the ontology we define fits the OWL-Full profile. The ontology is used for inference on individuals stored in the knowledge base.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

Based on the necessities of the platform further changes have been made to the semantic framework. Events are implied to be only external events (coming from the external analytics developers) and can be a stream. However, to filter these events and to make all the connections so that the affected parties are notified, some conditions need to exist. Only when these conditions are met, the notifications are sent. What we need to associate with a VE (Product) is the type of event that can affect it. The point which truly connects an Event Conditions with a VE Product is the Event Type, which is the first needed match. That type of event can affect something only when some conditions are met. So we use the idea of an "Event Condition" for this case. The basic idea is to have the "Event Condition" be the filter, however this is a very general concept (e.g. Humidity), which has some defined states and parameters for that state. These parameters are formed in a way that they can be easily clustered. There also exists a specific instantiation of an event condition, which exists in space and time, so as to be able send notifications when an "EventCondition Instance" fits the parameters that affect a VE product (and its associated Thngs) and happens near the location in which the items are at the moment. The matching of an EventCondition Instance's Location and Time Instant with the area and time period that a Thng of that VE class is located is in effect the second criterion that has to be fulfilled for a notification to be produced.VE is equivalent to product concept from EVRYTHNG. For this reason we use the two terms interchangeably. "Thng" is a specific instance of a VE or product.

The Supply Chain lets us model not only the route of products, from factory to consumer, but all the stages a VE product goes through, thus acting also as a lifecycle management concept. Products may be even used to construct other products. For instance a factory is supplied with raw materials, which are essentially products in the same way the items produced by the factory are products. Moreover the Supply Chain, as a concept, follows a specific sequence of stages as a base pattern. First the product is out of production, then it is transported to retail, the end user buys it at retail and if it is recyclable then, ideally, it ends up in a recycle bin. To be able to detect any kind of abnormality, we need to know if a 'Thng' (or many 'Thngs') does not follow the stages that the product goes through normally. This is why the declaration of a Supply Chain Instance is important. The instance must have declared "Chain Stages" and their sequence as well as their location. Models predict events associated with a "Supply Chain Instance" based on historical data of Thngs transitioning from one stage to the next.

4.2.1.2 AffectUs OWL Ontology Domain

4.2.1.2.1 List of Semantics

We make connections between all the aforementioned ideas to create a graph. This graph is the basis upon which flows make semantic queries and the knowledge base processes them. Without it there is no structure to the triple store that the knowledge base is, nor is there inference. To be able to make and use complex rules on the fly, we model ideas with a Vocabulary and links between its elements. As such we defined an ontology domain to hold this information in a format that is portable and widely used in the semantic web, which is RDF turtle. We use that ontology in our triple store for inference and reasoning. The triples that make the ontology are listed in the table below.

Table 3: Semantic concepts included in the AffectUs Domain

Subject	Predicat	Object	Comments

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

	e		
Event Condition	hasState	EventCondition State	EventCondition represents a concept such as e.g. humidity. EventConditionState is a specific state e.g. Dry. Populated by Ext Dev UI
Event Condition Instance	has Location Time	LocationTime	Indicates an occurrence of an event. LocationTime connects Location and Time instant into one entity. Populated by the events2KB flow
LocationTime	hasTime Instant	Instant (Time)	Reused existing time concepts from w3.org time ontology. Definition of class needed in previous line
LocationTime	has Location	Location	Location is a superclass of GeoJSON-DL vocabulary entities. Location is therefore stored as Geo-JSON.
Event Condition Instance	Subclass Of	EventCondition	Instance of an EventCondition. Populated by the events2KB flow
Event Condition	isForTyp e	EventType	EventType is one of the following: Humidity, Luminocity, Precipitation, SocialNetwork, Temperature, Weather etc.). It is set to the event condition by the external developer. Populated through the Ext Dev UI
VE (EVRYTHNG Product)	isAffecte d By	EventType	A VE is in general affected by an EventType e.g. Humidity. Inference is made from the EventConditionStates that have a similar location and time parameters with the product instance. Populated in the Product Owner UI
SupplyChain Instance	predicti onsMad eBy	Model	Necessary for referencing the model.
Event Condition State Parameter	has MinValu e	Numeric	Optional field in order for the Ext Dev to declare how states are identified. This can be helpful for a consumer of the event to potentially include interest in only a subset of the detected states. Included for future use. Populated by the JSON state declaration in the Ext Dev UI
Event Condition State	hasState Paramet er	EventCondition StateParameter	This is necessary to be described by the Ext Dev in the respective UI in order to let the framework (and consumers) know the potential states of the detected event.
Event Condition State Parameter	hasMax Value	Numeric	Same as above for the hasMinValue case
Event Condition	isAscend ing	Boolean	Implies that higher is better. Populated by a button in the Ext Dev UI. Can be used in the future for selecting only specific states based on context.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

Event Condition	published At	Endpoint concept, where one could obtain the information on the appearance of such an event	Necessary by AffectUs in order to consume the respective notifications from the producer (Ext Dev) side. Populated in the Ext Dev UI
Producer	produces	Product (VE)	Who produces a product. Populated by the Product Owner UI, following the retrieval of the product IDs list from TIS/EVRYTHNG
Consumer	consumes	Product (VE)	Entity that consumes a product (or the respective thngs). Populated indirectly through the incorporation of the dependent input product as part of the supply chain model (App Dev UI)
Actor	has Endpoint	Endpoint	To be used for notification purposes, especially for the case of output notifications towards product consumers. Populated through the Product Owner UI. Endpoint includes the protocol, path and potential credentials needed for accessing that endpoint.
Chainstage	belongs To	SupplyChain Instance	Stage entity instantiated for this chainstage to be used for modelling and illegal state change events detection. Populated in the App Dev UI (Create Model)
SupplyChain Instance	subclass Of	SupplyChain	A differentiation between general concept and specific entity, populated by inclusion of name in the App Dev UI (Create Model)
SupplyChain	hasStage	Stage Enumeration	Supply Chain (as concept, or draft for a new "Instance") has Stages. Populated in the App Dev UI (Create Model)
ChainStage	isOfType	Stage Enumeration	Out of production, in transit, at retail etc. Used in the illegal state detection event. Populated in the App Dev UI (Create Model)
Chainstage	has Location	Location	Location that conform to the Geo-JSON standard. Necessary to map incoming scan data to stage type. Populated in the App Dev UI (Create Model)
ChainStage	has Previous	ChainStage	Necessary to indicate the proper sequence of stages (dictated also by the need for out of sequence scans mentioned in Section 2.1 and Section 4.4 detection)
ChainStage	hasNext	ChainStage	Same as above
Chainstage	has Average Duration Time	Duration	This is populated from queries to the historical data and then the results stored in the KB for this supply chain and chainstage. During runtime upon rule launch, we infer the limit and this is put on the runtime rule.
Chainstage	hasInputs	VE (Product)	ChainStage can have one or many products as inputs. Necessary for inferring usage and dependencies. Populated in the App Dev UI (Create Model)

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

Chainstage	hasOutputs	VE (Product)	ChainStage can have one or many products as outputs. Outputs may be different from inputs. Populated in the App Dev UI (Create Model)
Consumer	orders	COLLECTION ID (String)	This is necessary in order to have information about which batch is ordered by which entity.
Collection	consistsOf	Thng	Collection is made of many Thngs (items). Retrieved from EVRYTHNG API
Event Condition	isDerived From	(URI) Data Source	A link to another source (external) from which the EventCondition is Derived From. Populated in the Ext Dev UI, can be used to filter events based on wanted input sources (e.g. weather data from a specific more reliable source)

4.2.1.2.2 Overview of Ontology as a graph

All the subject-predicate-object triples listed at the table in 4.2.1.2.1 form a graph, with the subject and object as nodes, and the predicates as edges. In RDF/OWL triples are known as object-property-value, but they are a graph just the same. The difference with OWL ontologies however is that nodes are classes and each property (edge) has domain and range as subject and object. That is because the ontology is the metadata for our KB. Each entry in the KB is either an individual, a member of a class, or data associated with one.

Practically everything related to the semantic web is a graph. The individuals that are put in the knowledge base follow, at least to some degree, the structure that the ontology graph dictates. For that we showcase the integral parts of the ontology as a graph. For convenience, we use the VOWL standard for visualizing the Ontology.

First we examine the part of the ontology that has to do with Event Conditions. In this part we see the structure that dictates the filtering of events. The Event Condition Instance is needed for separating the rules of the Event Condition from the Instance that may fulfil some of them at a specific place and time.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

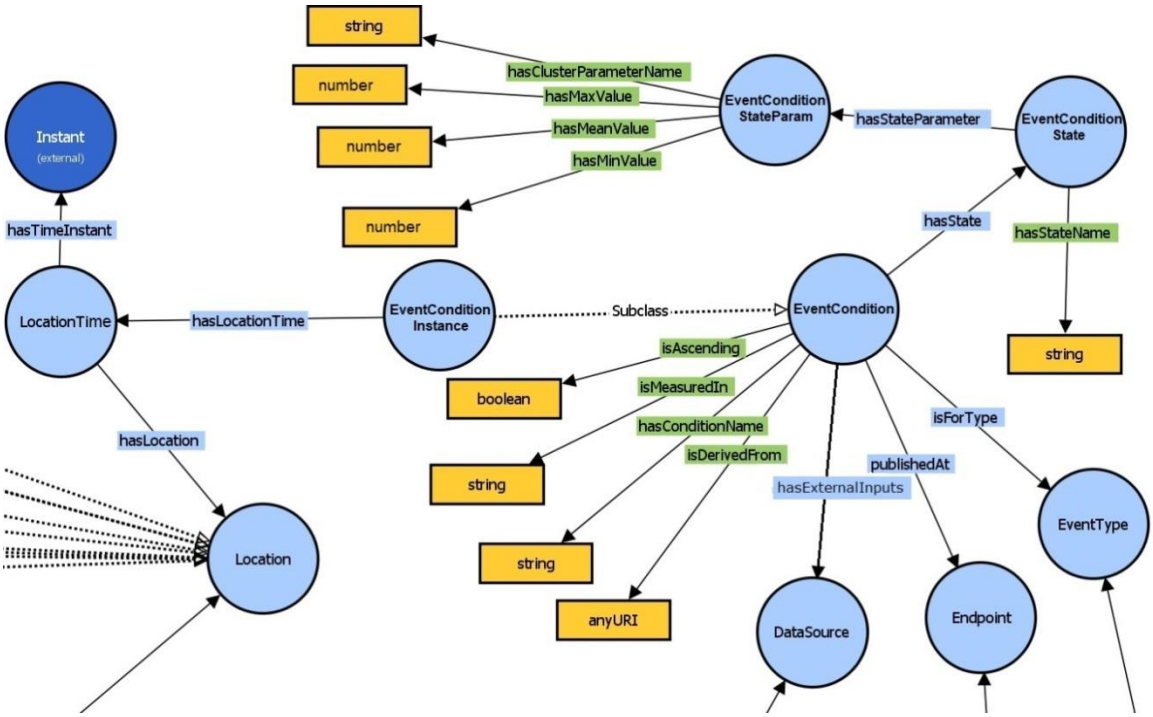


Figure 32: Ontology Part 1: EventCondition and Related Classes

Next we co-locate the parts of the ontology that have to do with the VE Product and the Supply Chain Lifecycle. We can also see how an Actor class can be identified as either Producer or Consumer. We also see AffectUs:Thng[1] as an instantiation of a product (in equivalence to the EVRYTHNG platform's representation), which is why it is declared as a Subclass. The Supply Chain is only made as a general concept linked to DBpedia's equivalent class, and a template for the Instances.

[1]Important note: Don't confuse owl:Thing with affectus:Thng , as owl:Thing is superclass to all classes in OWL.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

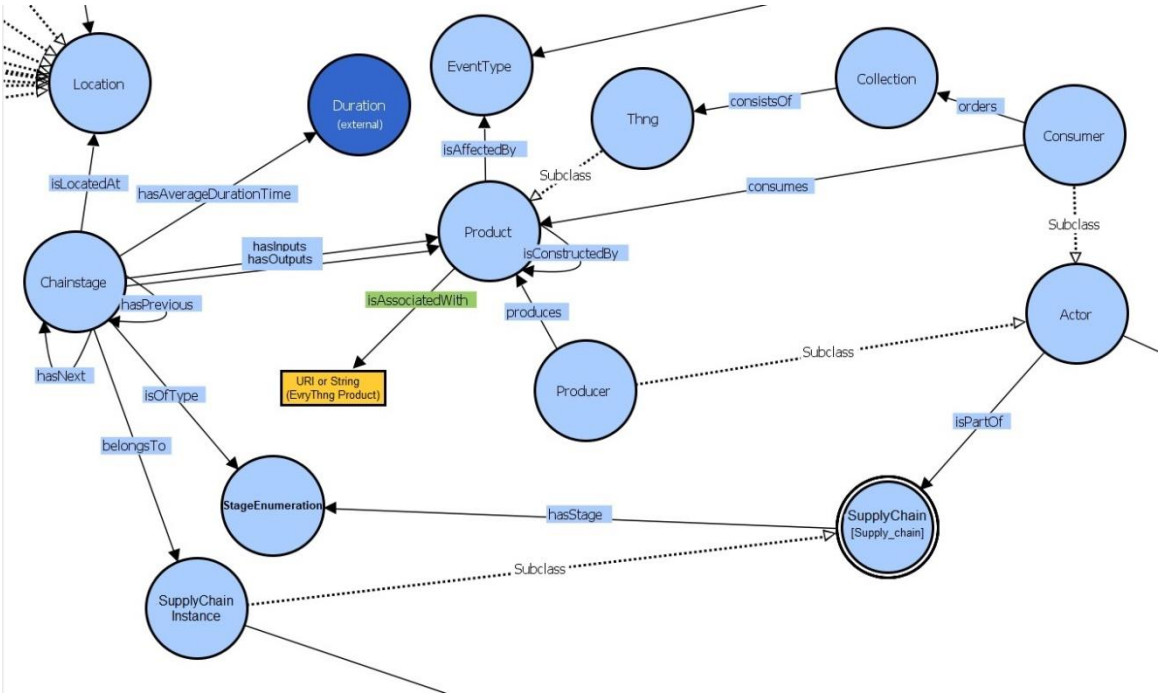


Figure 33: Ontology Part 2: Supply Chain and Product

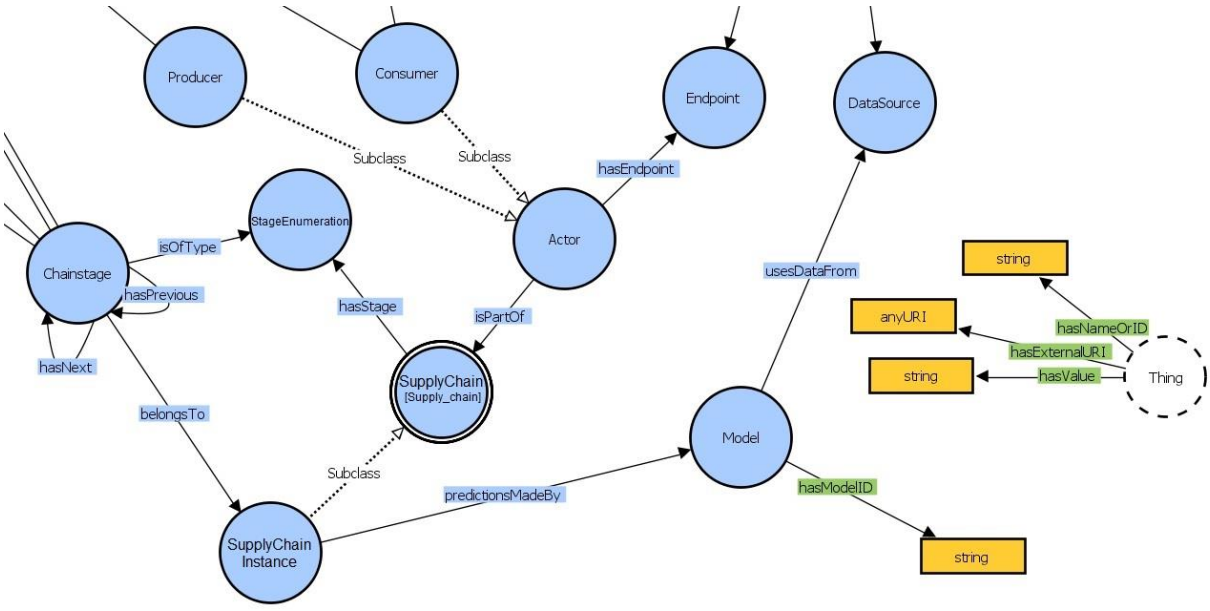


Figure 34: Ontology Part 3: Supply Chain and the rest

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

Finally we see the Location and Time instant concepts and how the affectUs classes use classes from external ontologies. Location follows Geo-JSON-DL standard.

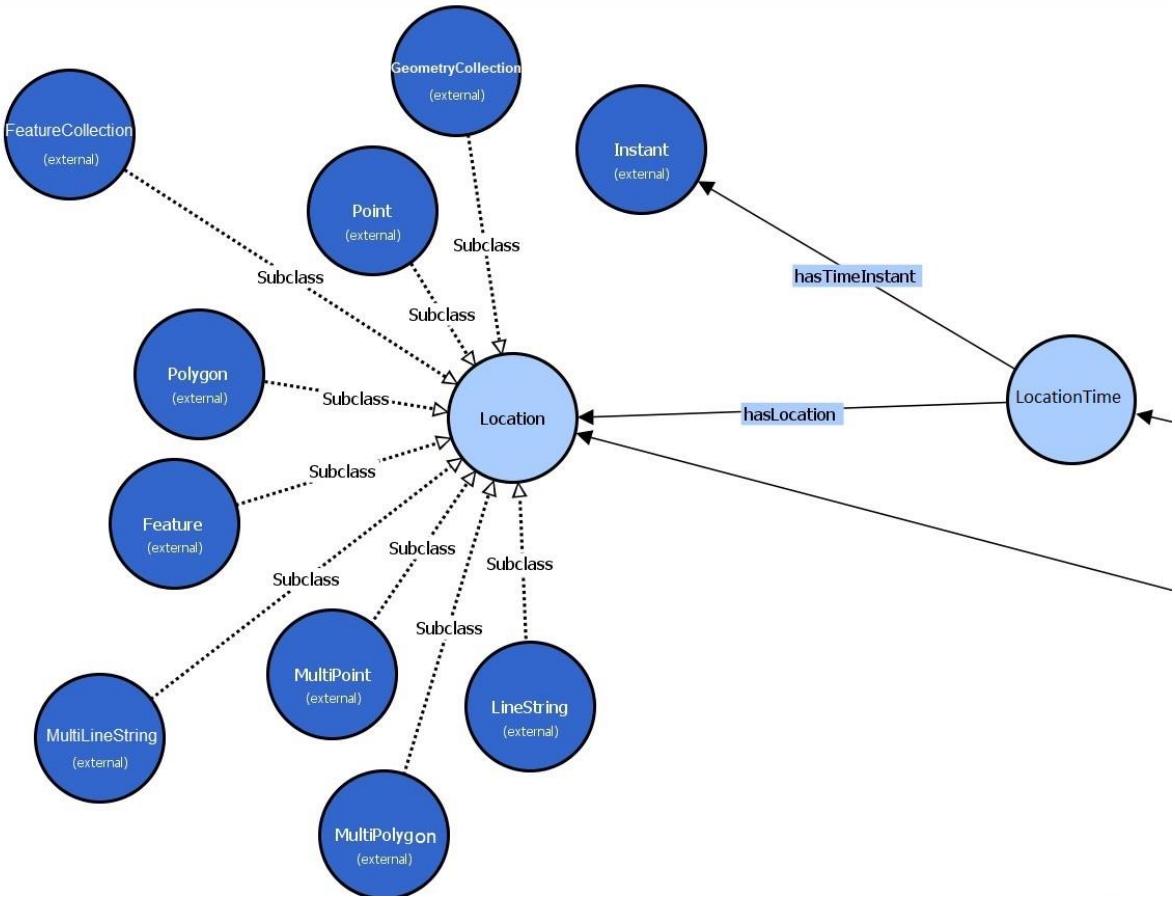


Figure 35: Example of Linked Data usage in AffectUs

4.2.2 Semantic Framework structure and implementation

The semantic framework is built on three fronts: The Knowledge Base which uses the OWL ontology for inference and reasoning, the SPARQL 1.1 entailed queries and the integration with programmatic rules on Node-RED side.

4.2.2.1 Implementation Details

The semantic framework in our case is made with a different approach from the usual semantic web implementations. Our architecture is service-oriented, therefore the KB is actually a Semantic Web Service. For this we use the Apache Jena Fuseki Server, which serves RDF data over HTTP. Fuseki uses SPARQL to communicate knowledge in RDF form. Normally Fuseki is only an RDF triple store with no reasoning capability over the data, however the inference support that Jena framework provides is available for use in the server. To do this we specify the Jena classes that Fuseki will use to set up reasoning.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

4.2.2.1.1 Configuring Fuseki as a Semantic Web Service

Regarding Fuseki as a KB, there are the following requirements:

- Isolate ontology from triples of individuals stored to KB by the UIs/Flows. In a sense it means isolating metadata from data. The terminological statements are secure and immutable, while the facts are updated constantly.
- We need to extract knowledge hidden in the inter-vertical connections between all entities in the system. With this functionality we are able to extract facts that are unknown before triple insertion. This is a very complex problem, which is why we use the Ontology as a set of complex rules for Reasoning/Inference upon triples of Individuals.
- To support knowledge extraction by using semantic queries, we need a structured language that fits the profile. SPARQL is not capable of that by default, as it is mostly a Query Language and a Protocol. However, by specifying a more complex entailment regime on the SPARQL queries, based on reasoning upon our ontology, relatively simple queries can return inferred knowledge that neither the users or queries specifically request.

The Fuseki server does not require writing Java code, like it is normally done with the Jena framework, or any form of hard-coding. It relies instead on a subsystem that uses Jena's Java classes to "assemble" the Services required. This subsystem is known as Jena Assembler. To give directions on how to create the services required, the Assembler uses RDF as configuration. In that RDF configuration file the triples act as directives to the assembler on what to use to build a service and its inner working. This is, in some sense, the semantic web equivalent of dependency injection. Jena provides the needed triple DB, or TDB in short, for doing CRUD operations on triples by using SPARQL 1.1 Request and Update. With Jena framework we also get access to reasoners, so that we can enable inference by applying a different entailment regime on the queries. To build the entailment regime that suits our needs we specify both the reasoner and the inference model it integrates. Essentially the "content" that the inference model uses is none other than the OWL ontology we created. The reasoner is Rules-Based and it constructs the rules internally, by using the content of the ontology. The collection of facts that the reasoner enacts reasoning upon is called "Base Model" and it is none other than the graph of individuals. Having considered all the above we have created a Fuseki configuration file with the triples that enable the needed functionality. The vocabularies used for such a configuration file other than the common rdf and rdfs, are ja for Jena Assembler, tdb for Tripple Database, and fuseki.

So the ontology is the terminological component, or Tbox, while the data added to the KB by SPARQL 1.1 are the assertion component, or the Abox. The format generally used for RDF triples is the "turtle syntax" or TTL which is the most common, and more easily readable than regular RDF/XML. SPARQL syntax is also based upon TTL.

```
@prefix :      <http://base/#> .
@prefix tdb:   <http://jena.hpl.hp.com/2008/tdb#> .
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ja:    <http://jena.hpl.hp.com/2005/11/Assembler#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix fuseki: <http://jena.apache.org/fuseki#> .

#Service Description
:service_tdb_all  a                             fuseki:Service ;
        rdfs:label                              "TDB affectus" ;
        fuseki:dataset                          :tdb_dataset_readwrite ;
        fuseki:name                             "affectus" ;
        fuseki:serviceQuery                     "query" , "sparql" ;
        fuseki:serviceReadGraphStore            "get" ;
        fuseki:serviceReadWriteGraphStore       "data" ;
        fuseki:serviceUpdate                    "update" ;
        fuseki:serviceUpload                    "upload" .

#Individuals Graph
:dataset a              ja:RDFDataset ;
        ja:defaultGraph <#model_inf> ;
        .

#Inference Model
<#model_inf> a ja:InfModel ;
        ja:baseModel <#graph> ;
        #Reference to model.ttl file, which is our ontology
        ja:content [ ja:externalContent <file:////home/affectus/apache-jena-fuseki/run/databases/model.ttl> ] ;
        #OWL BASED REASONER
        ja:reasoner [ ja:reasonerURL <http://jena.hpl.hp.com/2003/OWLFBRuleReasoner> ]
        .

#Tripple Database
<#graph> rdf:type tdb:GraphTDB ;
        tdb:dataset :tdb_dataset_readwrite .

#Where the TDB is located.
:tdb_dataset_readwrite        a             tdb:DatasetTDB ;
        tdb:location  "/home/affectus/apache-jena-fuseki/run/databases/affectus" ;
        .
```

Figure 36: Jena Fuseki Server Configuration

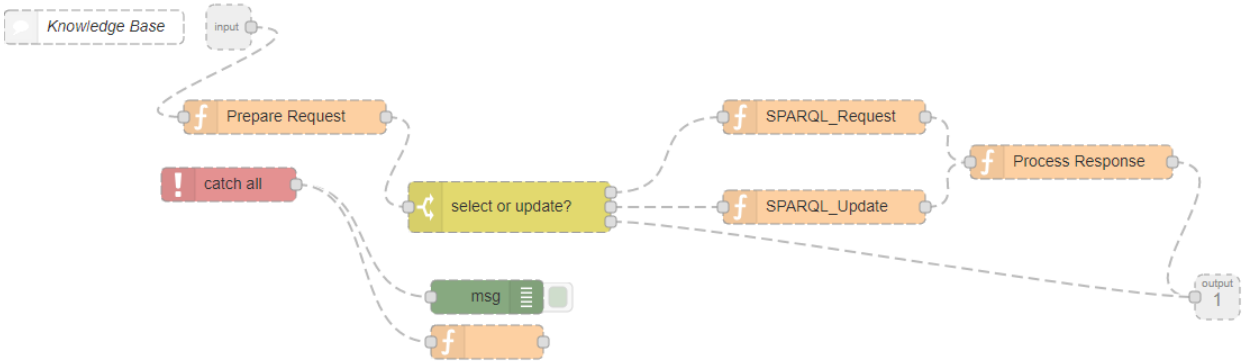
The service we create has a different endpoint for each operation. Queries are done in '/query', and update in '/update' endpoint. There are also some other services available for management through the Fuseki server graphical interface, such as uploading a dataset.

4.2.2.1.2 SPARQL and the OWL 2 RDF-Based Semantics Entailment Regime

In SPARQL 1.1 there exists the notion of entailment regimes. An entailment regime specifies how an entailment relation, such as RDF Schema entailment, can be used to redefine the evaluation of basic graph patterns from a SPARQL query making use of SPARQL's extension point for basic graph pattern matching. By default SPARQL 1.1 defines the evaluation of a basic graph pattern by means of subgraph matching. This means that when a query is issued, only the subgraph that strictly matches the query is returned. This is why, when we use SPARQL, we use it based on an entailment regime, so as to get richer and better structured results, that are produced by inference models. We use our ontology with OWL 2 RDF-Based Semantics Entailment Regime to accomplish the goal of finding more knowledge in each query. The configuration we showed in 4.2.2.1.1 enables this regime, so that the semantic queries functionality is available for the Node-RED flows.

4.2.2.1.3 Integration with Node-RED and Rules

To create this semantic communication between flows and the Semantic web service we create a connector which enables querying and processing the KB at will. We have to note that there is no Node-RED node readily available to accomplish that, however we create a sub-flow that is then incorporated in the respective flows. We configure a SPARQL over HTTP connector inside the sub-flow nodes to handle SPARQL 1.1 Query and SPARQL 1.1 Update. The formulation of these queries is done directly from JSON with SPARQL.js, and the response is then processed. In the query case the response is JSON, with the variables specified in the SPARQL query as parameters, and in the case of update the response is only either "success" or "failure" and the reason for it.



Knowledge Base Communication Subflow

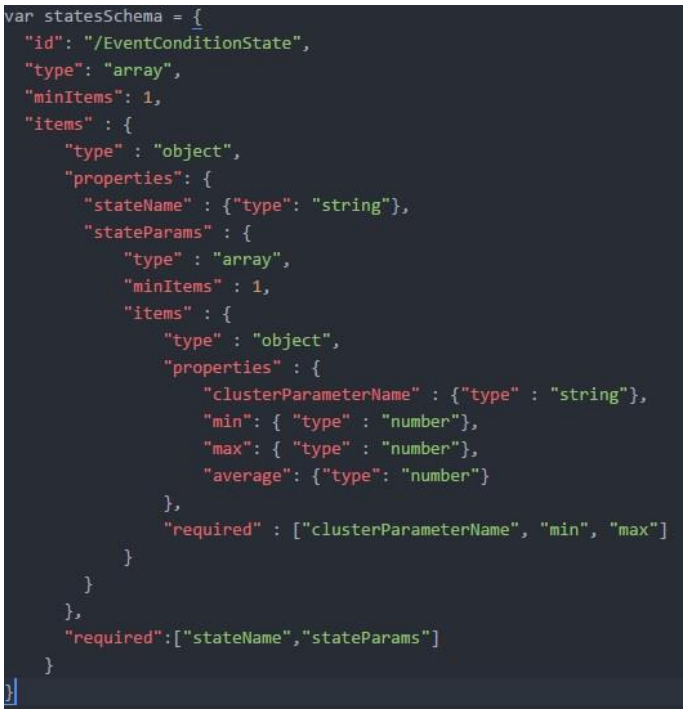
This sub-flow is called by all the flows, sub-flows or nodes necessary. There are fundamental differences between Request and Update, not just in Syntax, but also the way the communication is done (SPARQL over HTTP GET and POST respectively). Finally preprocessing and post-processing of the requests are necessary to ensure integrity.

4.2.2.1.4 Alternative rules and integrity checks.

In each use case of the Semantic Queries, there are times that further rules have to be applied that are flow specific. The way the SPARQL response is formulated, which is JSON-LD is so that we can easily make further processing without adding load to the graph. While the fuseki server does its own integrity checks based on the ontology, there are rules that need to be enforced on the Node-Red Side. In this case we use JSON-Schema to filter the data flow, or to make rules based on queried knowledge from the KB. Extra case-specific rules and integrity checks are the only logic that is implemented in Node-Red, while the rest is done by the reasoner on the KB side.

An example of Node-RED side validation is the Event Condition States Schema:

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.



```
var statesSchema = {
  "id": "/EventConditionState",
  "type": "array",
  "minItems": 1,
  "items" : {
      "type" : "object",
      "properties": {
        "stateName" : {"type": "string"},
        "stateParams" : {
            "type" : "array",
            "minItems" : 1,
            "items" : {
                "type" : "object",
                "properties" : {
                    "clusterParameterName" : {"type" : "string"},
                    "min": { "type" : "number"},
                    "max": { "type" : "number"},
                    "average": {"type": "number"}
                },
                "required" : ["clusterParameterName", "min", "max"]
            }
        }
      },
      "required":["stateName","stateParams"]
    }
}
```

Figure 37: The Event Condition States schema for the validator

4.3 Prediction Framework

4.3.1 Description of available prediction methods

4.3.1.1 Supply Chain Transition modelling

4.3.1.1.1 Generic description

The modelling process is about creating a prediction model for the transition between the stages of a production chain, which includes also indirectly the various stages through which a VE instance moves during its lifetime. Its main targets are two; (a) to ensure that all transitions follow a clearly defined order and (b) to identify delays in the transitions. The first target is tackled by the connected ontology, that clearly defines the allowed order that a thng has to follow in order to traverse through the stages chain. This order can be simple or it can have loops, depending on the product type and the chain architecture. In any case, the model is responsible for identifying violation in this order of transition and notifying the responsible parties with a new event.

The other target is handled by the model, which creates a prediction for the expected duration that a thing should stay in a specific stage of the chain. If the thing stays in this stage more than the expected duration, this could mean that something is malfunctioning or that the chain needs to be redesigned. Either way, an alert that a thing is being delayed at a specific stage would be useful to the interested parties.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

4.3.1.1.2 Data Inputs Templates

The input data to the model contain the following elements:

product_id: The id of the product type (VE) that the targeted thing is an instance of.

thing_id: The thing id for the targeted thing.

data: A JSON array that contains the data to be processed by the model. Each row of this array contains a set of features to be used by the model and the target feature, which is the duration in our case. The duration can be either null or numeric in the input data. If it is null the model will return the predicted duration for the features provided. If it is a numeral, the model will return the predicted duration for the features provided but then it will use this duration for self-evaluation and then retraining if needed.

A detailed look at the expected input and some examples are present in Section 4.3.2.2.

4.3.1.1.3 Data storage

The data storage used for this model is a MongoDB. The collection used contains a cache of the most recent data requests containing the duration value, that the model is using for training and evaluation functions. When a new input is registered, containing a numeric value for the duration feature, the oldest record in the collection is deleted and the new one is inserted, keeping the cache updated in case the model needs to retrain itself.

4.3.2 Prediction Framework structure and implementation

4.3.2.1 Implementation Details

The prediction framework is developed in Python, using the Google's TensorFlow library [19]. The data are converted to Tensors using Node-RED and then passed to the python backend through a Restful web service API, which in turn returns a prediction about the expected duration that a thing should spend in its current stage. In order to make that prediction we are employing a Deep Convolutional Neural Networks (DCNN) model ([6]-[8]), to be trained on data extracted from TIS data and external sources, such as weather, LCC and traffic information, details for which are described earlier in this document.

The following figure presents the Node-RED frontend of the predictor. The functionality of this flow is listening to a web socket for prediction requests, formatting the received data into Tensors and then passing them to the Python backend in order to receive the prediction and retrain the model to the more recent data at the same time. For that, a sliding window is preserved, storing recent data for retraining the neural networks, implementing an online learning methodology.

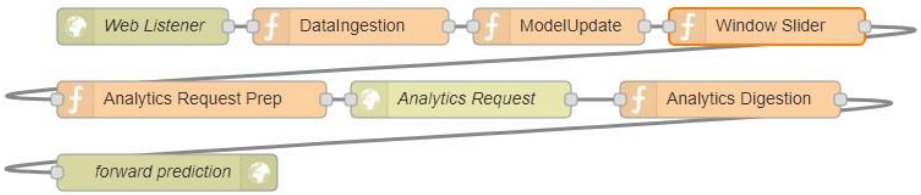


Figure 38: Prediction Modelling Node-RED flow

4.3.2.2 Information template

The prediction request is defined by a JSON array containing information about the things that the request is about. Each thing in the array contains its thing_id, the product_id that describes which product type (VE) the thing belongs to. In addition, it contains an array of data which detail the requested features to be used for the duration prediction. In the following example, we can see that each JSON object in the data array can contain a different number of elements, providing a considerable degree of flexibility in order to cover the various needs of different stages.

For example, we see that in order to predict the duration the thing will spend in the "in_transit" stage, the request is providing us with the timestamp that the thing entered this stage, the temperature, the wind, the humidity, the LCC and the traffic values, each of which will be used as a feature in the neural network model, along with the stage name, the thing_id and the product_id. On the other hand, when we are trying to predict the duration for the "at_retail" stage, we are only provided with the temperature and wind values, meaning that the rest of the features used in the previous prediction are not related to the duration a thing spends in a retail store.

If the duration value of a data point is not null then that means that the thing has already moved on from this stage, so the prediction is used only for evaluation purposes and if needed for retraining the model. For example, the third data point we see in the following example contains the value 3582 for the duration of the "at_customer" stage. This means that the thing spend 3582 milliseconds in this stage before moving on to the next one, as defined by the relevant ontology.

```
[{
"thing_id":202cb962ac59075b964b0715,
"product_id":c81e728d9d4c2f636f067f89,
"data":[
        {"timestamp":1515435312853,"chainstage":"in_transit ","temperature":16,"wind":30,
        "humidity":18.23,"LCC":64,"traffic":1642,"duration":null},
        {"timestamp":1515435320681,"chainstage":"at_retail ","temperature":12,
        "wind":33,"duration":null},
        {"timestamp":1515435371872,"chainstage":"at_customer","duration":3582}
]
}]
```

Figure 39: JSON example of model input including external events

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

The prediction model flow has two modes; the RestFul API and the daemon. The input in both cases is the same and it is the one described in this section. The output on the other hand differs. In the case of the RestFul API the result is a prediction about the expected duration of the things specified by the input data. In the daemon case, the flow creates and sends a new event, signaling the delayed status of a thing. The event is described by the following structure:

```
{
"eventConditionname":"stage_change",
 "statename": "delayed",
 "Details": {
        "timestamp": value,
        "thing_id": value,
         "stage": value
}
}
```

Figure 40: Example of JSON publication for supply chain related event detection

4.3.2.3 Integration with TIS

This component is not directly integrated with TIS but it is using the already mentioned data which are derived from the TIS scans. These data include the stage that a thing currently is in, the time it entered this stage and the duration it spent in this stage.

4.3.2.4 Integration with other AffectUs Backend components

The chain stage modeling is using the ontology, through a helper flow, in order to enrich the TIS data with semantic information, regarding the stage that the scan concerns and the next and previous stages.

The predictions about the expected stage duration are also useful to other components, such as the chain history interface and the chain monitor interface, which split the things that belong to each stage to normal and delayed, comparing the expected duration with the actual duration of the thing crossing the stage.

In the general use case, described in D1.1, the chain stage model relates to the Create Model box, as show in the following figure:

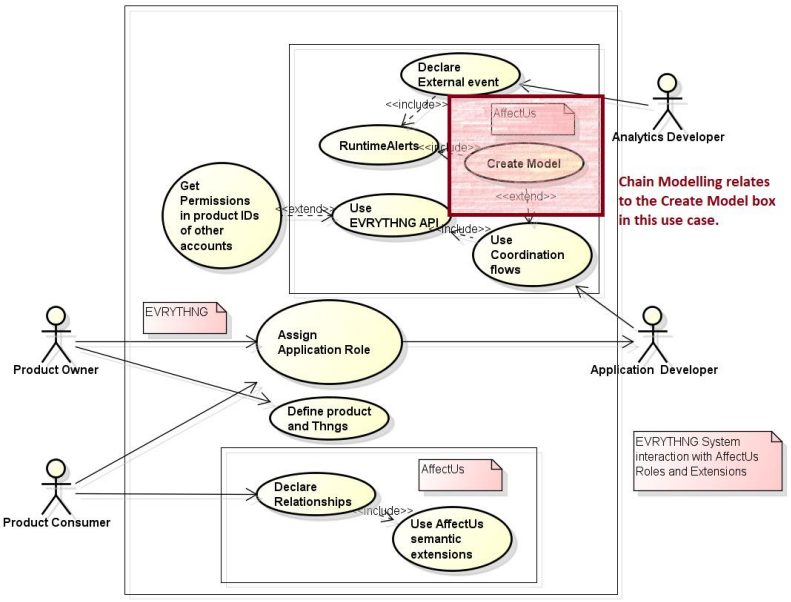


Figure 41: Chain Model place in the general use case diagram

4.3.2.5 Other Helper flows

The stage model is using a number of helper flows, as already mentioned. It is using a helper flow that enriches the TIS scans with semantic information, using the relevant ontology, described in Section 4.4.1. This is needed because the TIS scans do contain locations but we still need to correlate these locations with a specific stage type. Another helper flow is collecting the data from all relevant sources, storing and updating the training data set in a MongoDB collection.

The third helper flow used by the stage modelling is periodically making requests to the model in order to keep the predictions updated and present a valid representation of the expected behavior of things traversing through their production chains. This third flow is enabling the prediction model to run as a daemon and monitor the chain activity for delays, sending the appropriate events as defined earlier to the interested parties, as well as updating the records in the database.

4.3.3 Usage

As mentioned the model has a RestFul web service API so a user just needs to send a request to that API, containing the information defined earlier on in this chapter. Then the model will process the request and return a JSON array, containing all the predictions for the data included in the original request.

Endpoint	/chain_model
Input Type	JSON Array as described in chapter 4.3.2.2, in which each row is a different prediction request.
Output Type	JSON Array containing one response per input array row.
Method	POST
Input Field Name	“data”

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

4.3.4 Future Work (Model Creation UI)

To continue this work we plan on creating a user interface that enables the users to create customized models, using the Python backend. The user will be able to provide JSON arrays of data, containing features and a numeric prediction target. These data will be automatically and effortlessly converted to Tensors by the interface, forwarded to the python backend where a new DCNN model will be trained. The model will be saved and then provided to the user for later usage.

The interface will also provide evaluation options that allow the users to test their models, deciding if their current choice of features is appropriate for their prediction problem. This real time fiddling is expected to provide the users with a means of understanding the effects of each feature to the prediction model as well as testing the effectiveness of the DCNN models on a number of different prediction problems.

4.4 Linking/Integration Runtime layer

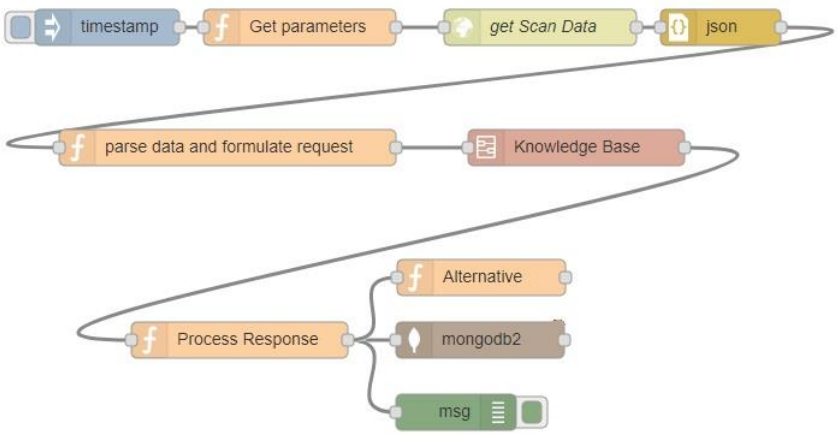
4.4.1 Scan data ingestion and transformation

Scanned data received from the EVRYTHNG platform undergo a dual process including:

- Transforming their location to be mapped to a “ChainStage”, one of the stages of the supply chain, a process that requires querying the KB for determining in which stage this location relates to.
- Ingesting this form into the MongoDB instance in order to be used as historical data for the Prediction Framework modelling

When a product is scanned, the data is parsed, and for each scan, we keep the scan properties the time and the location. After that the location is mapped to the corresponding stage, by querying the KB for ChainStage by Product and Location. The results from KB have the chainstages matched to each scan. After that the data is inserted into MongoDB as historical data and at the same time is set to be readily available for short-term usage after that. The purpose is to have the data ready to be forwarded in the next stages, at the runtime layer.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.



Scan Data Ingestion Flow

Figure 42: Scan data ingestion flow

4.4.2 External Event Publication Template definition

As indicated in Section 4.1.6, the external analytics developer needs to upload the schema based on which the event notifications will be published. Even though the states declaration is up to the developer, a number of other fields are deemed as obligatory, as indicated in the following JSON example. Such information includes the name of the event, along with a details object describing the state (can be more complex than the example) and definitely the time and location (in GEOJSON) of the event. Especially the last two are critical since correlation between an affected product (based on the generic category declaration) will be secondary performed via the proximity in time and space.

```
{
        "eventConditionname":"humidity",
        "details": {
        "statename": "dry",
        "timestamp": 5,
        "location": GEOJson
        }
}
```

Figure 43: JSON example of event publication

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

4.4.3 Specific Event detection logic (Abnormal Sequence of States)

There are cases in which the supply chain a product belongs to, has to be checked for abnormalities or illegal sequence. These cases are viewed in section 2.1, and are triggered by scanning data arrival and processing. In the end of flow in scanning data ingestion (section 4.4.1.1) the data is kept temporarily in the runtime layer's context. One of the reasons it is done is because we need to keep data about the previous stage of a Thng. We keep this data because it enables us to determine whether the transition between stages is legal or not. So, another flow is triggered that enforces the correct sequence. The basic stage sequence, which is recommended to be followed when registering a supply chain, is:

"OutOfProduction"-> "In Transit" -> "At Retail" -> "At End User" -> "At Recycle Bin"

Each registered supply chain instance follows a variation of the pattern, and it is kept in the KB. If the ChainStage of the scan is the same as the ChainStage registered as next in the data kept from the processing of the previous scan, then then the flow does not continue and no further action is taken. If however this is not the case, then we continue by checking the KB for extra Data. If the data indicates that there is an error in the sequence, then we trigger the notification.

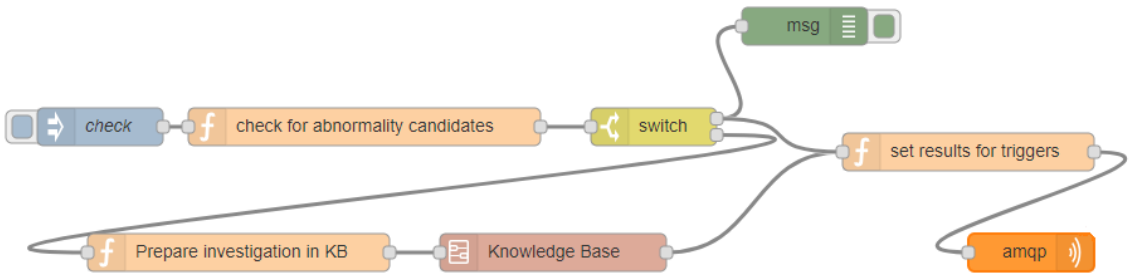


Figure 44: Abnormal Sequence flow

For each case the KB is queried to return knowledge that indicates the validity of the error, and the connections around it. The KB returns the connected entities which are either used for setting up the right triggers or to provide complementary information.

The flow is checking if the previous, the current and the next stages of the incoming data are consistent with the ones registered in the ontology and checks if it can confirm the previous stage from the data in the context. If at least one of these conditions is erroneous a new event is created and sent to the interested parties. The possible events have the following structure, depending on which factor is erroneous:

```
{
```

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

```
“eventConditionName”:”stage_change”,
 “statename”: “skip”,
 “Details“: {
            “timestamp”: value,
            “thing_id”: value,
            “prev_stage”: value,
              “new_stage”: value
            }
}
```

The skip event is created when a thing is bypassing, or skipping, one or more stages.

```
{
“eventConditionName”:”stage_change”,
 “statename”: “counterfeit”,
 “Details“: {
            “timestamp”: value,
            “thing_id”: value,
             “prev_stage”: value,
             “new_stage”: value
            }
}
```

The counterfeit event is candidate creation for when the stage is “Out of Factory” or later, if the Actors associated with it are different than who they were, or undefined while in the previous they were OR when the location of the same stage has changed. If the same Thng exists in different Stages, and these are parallel in the Supply Chain Instance (one cannot be a next stage of the other), then the counterfeit event is created.

```
{
“eventConditionName”:”stage_change”,
 “statename”: “reverse”,
 “Details“: {
            “timestamp”: value,
            “thing_id”: value,
            “prev_stage”: value,
            “new_stage”: value
            }
}
```

The reverse event is created when a Thng is trying to enter a stage that it has already completed. This can be allowed in some cases, such as when a thing is returned to the sender due to a malfunction or a mistaken order. In most cases though, it is suspicious and the interested parties should be notified.

4.4.4 Messaging Logic

The messaging logic in AffectUs can be implemented in two ways. The first one, in Section 4.4.4.1, includes a common publication point, which passes through the KB to indicate the dependencies and necessary outputs to which the notification should be redirected. This is simpler yet it is expected to

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

create a bottleneck in the KB querying. The second case relates to a future plan to improve the scalability of the system by embedding the messaging setup logic in the AMQP access model, as indicated in Section 4.4.4.2.

4.4.4.1 Event Receipt and Notification Output Abstraction Layer for Endpoints

AffectUs supports the redirect of information from its main messaging system to a number of endpoints for the adaptable notification to the affected entities such as Product Owners. For this reason the consuming entities need to declare the type and path of the endpoint, information which is stored to the AffectUs system for future use, as indicated in Section 4.1.2.1.

Initially an AMQP exchange may be created, in which all the publishing entities will forward their information, both external developers and chain models. Inside Node-RED, a relevant abstraction layer is created that adapts the outputs of its messaging system to the specific endpoint of each affected entity.

This layer is used in order to push notifications from events that are deemed to affect this specific user based on the declared or inferred dependencies. This is performed via the flow indicated in Figure 45. Events notifications arrive from the main AffectUs publication endpoint, triggering a semantic query to the KB in order to discover affected entities and their declared endpoints. For each of these entities the message is adapted to the relevant node implementation and way of node configuration and the datum is sent in the respective branch. The list of supported protocols (currently HTTP POST, AMQP ,MQTT and NMA) can be extended by inclusion of a) a new option in the protocol enumeration list of Section 4.1.2.1 b) the creation of a new branch with the respective protocol implementation node in the flow of Figure 45.

The event is also stored in the KB for future use.

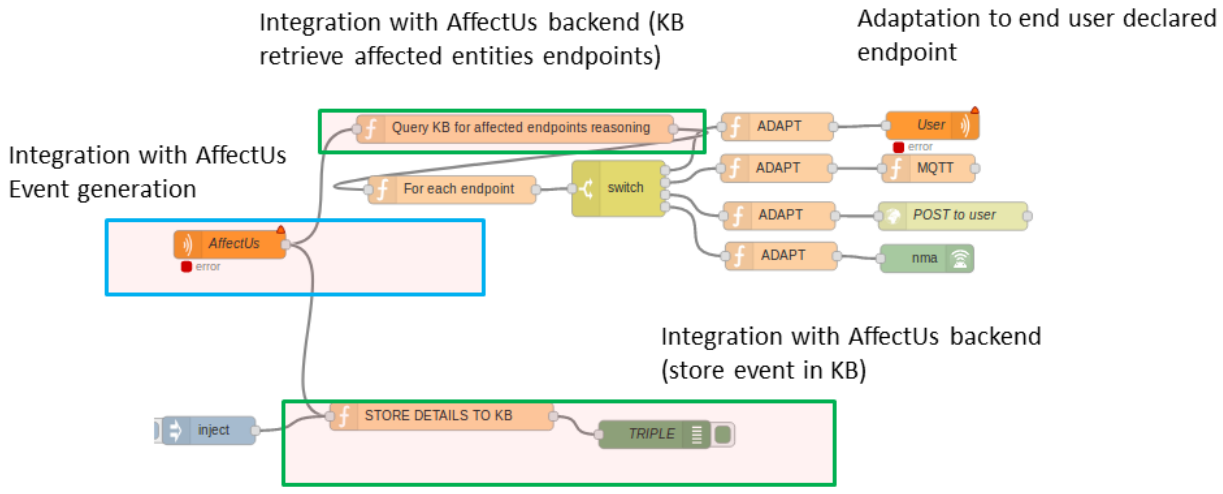


Figure 45: Structure of the Notification Abstraction Layer (Events_2_KB_flow)

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

4.4.4.2 AffectUs Messaging System setup for enhanced scalability

AffectUs utilizes the AMQP protocol for its internal messaging structure (and the respective RabbitMQ implementation of it). Therefore, if one maps specific roles to the rationale of AMQP resource creation (exchanges, bindings and queues), an optimization may be performed without the need to query the KB at each event arrival. In detail the following rationale may be followed

- For each external developer and event, a respective AMQP exchange needs to be created, with the naming convention “eventName”. The developer has write access in the specific exchange, in which they need to push the specific notifications for this event.
- Likewise, for each modelled product, a respective exchange needs to be created (with the chain or product name), in which the prediction framework of the supply chain will forward events related to that chain.
- For each listening entity in AffectUs, a relevant queue is created, in which the specific entity has read rights. In case a declaration is performed that links a product of this entity with a specific event, as indicated in the previous sections, a registration (binding) is performed in the messaging system linking the event’s publication exchange with the entity’s listening queue, therefore enabling the propagation of the respective notifications. A default binding is also performed between the product exchange and the owner’s queue.

The plan is to move towards this version in the future, which however needs changes to be incorporated in the existing flows.

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

5 Conclusions

Following the design considerations included in D1.1, this document aimed at implementing the necessary functionalities to address requirements expected from the AffectUs system. In Table 4 the Requirements Traceability Matrix from D1.1 is presented, annotated with an extra column to indicate the position in this document where the according implementation has taken under consideration the expressed requirements.

From this mapping it is evident that all requirements have been taken under consideration during the implementation stage. Functionalities have been abstracted based on implemented UIs, hiding the underlying complexity of the system, while flow programming rationale has helped in integrating multiple internal and external systems and APIs. Furthermore extensibility is easily achieved through the creation of extra flows and plug-in to the respective endpoints, for either receiving or publishing notifications.

Furthermore, through the selection of the implementation and packaging technologies (Node-RED and Docker), reusability of the provided components can be easily accomplished, without the need to understand the overall system complexity. Indicatively, system setup can be performed with a minimum set of Docker commands, while relevant Node-RED flows can be directly inserted in an existing Node-RED installation, in a copy and paste manner.

Future plans with relation to AffectUs relate to the usage of the provided extension in order to validate its functionality from external audiences, as well as extension of the list of external events that will enable a higher level of inference and cross-correlation of information coming from even more data sources in the context of Smart Cities.

Table 4: Updated Requirements Traceability Matrix Following Implementation and Mapping to Flows

Requirements	System UC (by name)	Component	Sequence	Priority	Implemented in flow (Section)
Affect_REQ1 (Inclusion of external events developers that offer generic notifications)	Declare External Events	According UI element, KB rule, messaging flow	Declare External event	SHOULD	AffectUs Ext Dev U for event declaration (4.1.6), Affect Us Product Owner UI for dependency declaration(4.1.2), OWL concepts for external events (4.2.1), Event distribution logic (4.4.3)

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

Affect_REQ2 (Shared IDs rationale between TIS platform and AffectUs)	Generic role interaction UC	KB, data processing, coordination flows	Declare relationships, Create model, Trigger event identification	MUST	Product Owner Enable Access for product retrieval (4.1.2), Storage in AffectUs Mongo DB based on IDs retrieved from EVRYTHNG (4.1.2), Product list population through the same source (4.1.2 for dependencies, 4.1.4 for chain modelling)
Affect_REQ3 (Abstract and adaptable supply chain analysis)	Create model	According UI element, Model Creator, KB structure, DB	Create Model	SHOULD	AffectUs App Dev Declare Chain UI (4.1.4), OWL structure (4.2.1)
Affect_REQ4 (UI insertion for dependencies)	Declare External Events, Declare Relationships	According UI elements, Reasoner, KB	Declare Relationships, Declare external event	SHOULD	AffectUs App Dev Declare Chain UI (4.1.4) for chain related dependencies, AffectUs Ext Dev UI for scope (category) of event declaration (4.1.6), Product Owner Enable Access "Declare Dependencies" tab for effect from scope (4.1.2)
Affect_REQ5 (Fine-grained micro-services approach)	N/A	All components	All sequences	COULD	Docker based system and component setup (Section 3)
Affect_REQ6 (smart tag "inner" sensor values consideration)	Receive runtime alerts	KB, coordination flows	Trigger Event identification	SHOULD	OWL structure in 4.2.1, Declaration of category from product owner in (4.1.2), Notification mechanism structure in 0. However in this case an overlap exists with

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

					the EVRYTHNG platform's ability to define such notifications based on value limits. For this reason we have decided to assume that such rules are setup in the EVRYTHNG environment, we are receiving them through proper registration and we are only forwarding them to the affected entities

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.

6 References

[1]. D1.1 AffectUs Deliverable Component and Experiment Design Report, October 2017

[2]. EVRYTHNG Roles and permissions: https://developers.evrythng.com/docs/roles-and-permissions

[3]. George Kousiouris, Adnan Akbar, Juan Sancho, Paula Ta-Shma, Alexandros Psychas, Dimosthenis Kyriazis, Theodora A. Varvarigou: An integrated information lifecycle management framework for exploiting social network data to identify dynamic large crowd concentration events in smart cities applications. Future Generation Comp. Syst. 78: 516-530 (2018)

[4]. GEOJSON format: http://geojson.org/

[5]. Base EVRYTHNG application roles setup: https://dashboard.evrythng.com/resources/userRoles/base_app_user

[6]. M. Abadiet al., "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," ArXivPrepr. ArXiv160304467, 2016.

[7]. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in neural information processing systems, 2012, pp. 1097–1105.

[8]. Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436–444, 2015.

[9]. "Convolutional neural network," Wikipedia. 09-Nov-2017.

[10]. COSMOS Get Twitter Data flow: http://flows.nodered.org/flow

The AffectUs project is a TagItSmart (Grant Number 688061) Open Call 1 Extension project co-funded by the European Union (EU) Horizon 2020 program.