A solution for processing supply chain events within ontology-based descriptions

Borja Ramis Ferrer, Wael M. Mohammed and Jose L. Martinez Lastra
FAST-Laboratory
Tampere University of Technology
PO Box 600, 33101 Tampere
{borja.ramisferrer, wael.mohammed, jose.lastra}@tut.fi

***Abstract*—The industry is constantly moving towards the research, implementation and deployment of new solutions that permit the optimization of processes. Nowadays, such solutions consist mostly on ICT developments, which permit the collection, distribution, integration, analysis, and manipulation of heterogeneous data. The under way Cloud Collaborative Manufacturing Networks (C2NET) project targets the development of cloud-enabled tools for supporting the SMEs supply network optimization of manufacturing and logistic assets. The C2NET solution includes the implementation of paradigms as e.g. cloud computing or service-oriented and event-driven architectures, used for wide data integration. Among its requirements, the C2NET platform needs a solution for catch, process and react to events triggered in different locations of collaborative manufacturing networks, which are endowed with devices that permits the integration of Cyber-Physical Systems (CPS). This paper presents the architecture and main functionality of a knowledge-based approach that allows processing supply chain events handled by CPS devices. The main component of the presented solution is the SECA ontology, which is a knowledge base that can be updated at runtime. The main purpose of the ontology is to describe events, their status and the actions to be performed once a set of events are triggered in certain order. This research work offers a solution that can be employed by the C2NET platform not only to catch and process events; but also to notify linked data consumers.**

Keywords—knowledge-based system, ontology, SPARQL, event processing, cyber-physical systems

I. INTRODUCTION

Nowadays, industrial enterprises need to manage and adapt to user demands, which accentuate more the dynamism and reconfiguration of contemporary industrial systems. Therefore, one of the priorities of the industry is to employ new ICT solutions that are capable of managing and analysing a vast amount of heterogeneous data, which is retrieved from different data sources. The customization of products and optimization of processes to enhance enterprise's efficiency are two factors that directly imply the need of exhaustive monitoring and control on events, which occur in different systems involved in supply chain product life-cycle.

The employment of ICT in the industry ends up in the implementation of different paradigms as cloud computing or Service-Oriented and Event-Driven architectures (SOA and EDA, respectively). Such different approaches can be used for wide data management, integration and distribution. For instance, Event-driven Service Oriented Architectures (event-driven SOAs) can be considered in manufacturing systems to be implemented and exploited within the actual generation of of programmable logic controllers [1] and Web Service (WS) enabled devices [2]. WS enabled industrial controllers (e.g. S1000 from Inico Technologies[1]) permit the implementation of SOA in actual production lines [2]–[4]. Basically, the use of SOA in manufacturing systems offers a robust solution for monitoring the status of industrial system resources and controlling processes. Such control on system's components permits the calculation of Key Performance Indicators (KPIs), which are indicators used to measure and, then, enhance the efficiency of industrial systems. Examples of standardized KPIs can be found in [5]. On the other hand, the integration of Cyber and Physical Systems (CPS) permits distributed, modular, flexible and reconfigurable solutions for industrial systems. Some of the challenges of CPS were firstly addressed in [6] and current advances presented in [7].

The European Commission stated that the research and consequent developments of ICT are mature enough to be implemented in enterprises, aiming e.g. the simplification of activities and improvement of communications between intra and inter systems [8]. This will benefit SMEs to overcome their constrained resources and make them more visible and active in collaborative networks, increasing their competitiveness.

In this context, the Cloud Collaborative Manufacturing Networks (C2NET) project[2] has the objective of creating cloud-enabled tools for supporting supply network optimization of SMEs' manufacturing and logistic assets, based on collaborative demand, production and delivery plan [9]. One of the requirements for creating the C2NET solution is the integration of different IT systems, which belong to participants or partners working in the same supply chain for sharing data to optimize overall processes in products' life-cycle. The C2NET project needs a solution for mapping the occurrence of certain sets of events to certain actions, which are monitored and/or controller by consumers (i.e. both internal and external systems from the C2NET platform).

[1] http://www.inicotech.com/
[2] http://c2net-project.eu/

The integration of EDA and knowledge-based systems within CPS permit the implementation of solutions to collect, distribute, integrate, analyse, and manipulate events (controlled by WS enabled devices), which occur in different locations of supply chains. In fact, this article presents an approach to implement such solution for processing supply chain events. Then, the manuscript describes the principles, architecture, used ontological model and proof of concept for implementing and testing the approach.

The rest of the paper is structured as follows: Section II describes the main research work related topics. Then, Section III presents the approach for creating a knowledge-based engine for processing supply events. Afterwards, Section IV shows a proof of concept and some results for validating the presented approach. Finally, Section V concludes the article.

II. Related Work

A. Supply chain events and data collection

Currently, with the support of computer and networks development, manufacturing systems term became wider and more comprehensive. As defined in ISA-95[3], the automation pyramid is constructed mainly by three layers, Enterprise Resource Planning (ERP), Manufacturing Execution System (MES) and shop floor [10]. One of the key terms is provided as supply chain in the ERP level. Supply chain represents the relation between suppliers, manufacturer and customers. ERP, which is managing the supply chain, bonds all three players of the supply chain. The definition of supply chain management comprises as a process for planning, implementing and controlling resources between the three main players in order to achieve the optimal results such as time efficiency and/or cost efficiency [11]–[13].

Therefore, supply chain events includes all the communication information between supply chain partners. The communication may involve negotiations, payments, orders, offers, supply and delivery planes. Due to the massive amount of events in supply chain, controlling and processing mechanisms are required.

Many projects tend to provide solutions for such problem. The C2NET project is creating a platform for supply chain management. The C2NET platform aims to provide a media for supply chain partners to communicate. One of the main development tasks is presented as Data Collection Framework (DCF). DCF allows the users to communicate with each other regardless the variations in the technologies which they use. A CEP (Complex Event Processing) engine is required for achieving such a requirement. Ordinary CEP engines provides an excellent result in terms of functionality. However, these CEP engines require a reasonable time for configurations and rule design as e.g. in [14] wherein users must define the events, status and methods based on Complex Event Detection and Response (CEDR) queries and its translation to concurrent reactive objects.

B. Knowledge-based systems

Within the appearance of ICT and web-based technologies, the development and implementation of knowledge-based systems is increasing. One of the common practices is to have a central knowledge base that can be accessed to either update or request information. Such repository of system information become a critical component for knowledge-driven solutions [15], [16]. Currently, aforementioned research works use semantic descriptions that are both machine and human readable. Then, it is easy to implement by humans and there are not problems for machines to reuse such knowledge. For instance, ontological models can firstly be designed by humans but afterwards be populated and manipulated by cyber systems.

Ontologies [17] are nowadays employed as a mean to describe knowledge of the system to be controlled, which is encapsulated as an engineering artifact that can be updated and consulted at operation runtime. Although there are several languages for designing ontologies [18], probably, the Ontology Web Language (OWL) [19] is the most famous one due to its level of description. OWL is an RDF-based language [20] that permits the representation of knowledge of any domain within different elements as e.g. individuals, classes, relations, restrictions, functions, axioms or rules.

As OWL is RDF-based, it can be queried within SPARQL Protocol and RDF Query Language (SPARQL), which permits to consult ontological data within different sorts of query types as e.g. ASK or SELECT, among others [21]. In addition, an extension of SPARQL, sometimes referred as SPARUL (SPARQL Update) allows to update RDF graphs with e.g. INSERT or DELETE query types [22].

Moreover, reasoning engines (or reasoners) are capable of analyzing ontologies and inferencing knowledge. Conceptually, reasoners are capable of concluding or creating new facts (implicit knowledge) out of stated facts (explicit knowledge). To assist such inference of knowledge, the definition of semantic rules is a common practice, within e.g. the Semantic Web Rule Language (SWRL) [23] for ontologies implemented with RDF-based languages. On the other hand, reasoning engines are also employed to validate ontologies so they are capable of identifying any type of model inconsistency.

III. Approach

This section presents the approach for developing a knowledge-based engine for processing supply chain events. As appears in Fig. 1, the proposed knowledge based engine may interact with various devices that runs on different platforms, as well as for consumers. On the other hand, the user may interact with the engine using web-based user interface for uploading the ontology.

Following subsections describe the principles, ontology model, architecture and interaction of components of the approach.

[3] https://isa-95.com/

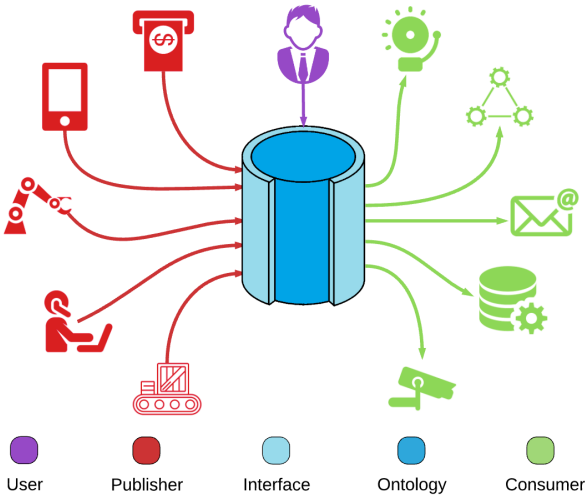

Fig. 1. View of the KB engine and interactions with a supply chain entities

A. *Processing events within a knowledge based engine*

The supply chain englobes all components and stages that participate in the development and shipment of products. Moreover, the manufacturing process starts from the management enterprise layer to the factory shopfloor. In event-driven environments, all manufacturing components connected to contemporary industrial controllers publish events in order to notify about status of machines.

In this research work, all events are considered as self-described events. This means that events are issued from the WS enabled device once a logic criterion is matched. For instance, a machine in the factory shopfloor measures the current continuously and if the current exceeds a certain limit an event will be issued, notifying to the interested event subscribers that the machine entered in the state of high current value. An example of a simple JSON formatted event to be published is shown in following Fig. 2.

```
{
   "eventId": "highCurrent",
   "publisherId": "robotA",
   "timestamp": 1462095503510
}
```

Fig. 2. An example of JSON event message

Such event format permits the low level of event processing to be occurred in the device, while higher-level to be achieved in the knowledge-based engine. In this way, the knowledge-based engine can state a high-level conclusion, which is defined by the occurrence of a set of events within a time window. Because the time is important to determine if the set of events occur in certain time window, the *timeStamp* attribute is included in the JSON message and will be taken into account in updating the KB of the engine, each time that an event is caught.

The ontology designed to handle such information and the manner how devices and the interface of the KB engine communicate is described in following subsections.

B. *The SECA ontology*

As previously described, the presented approach employs an ontology, which includes semantic descriptions of status and event-related information. The main ontology concepts are presented in Fig. 3. The model can be implemented with RDF-based languages as e.g. OWL and the queries used to interact with it are shown in next subsection.

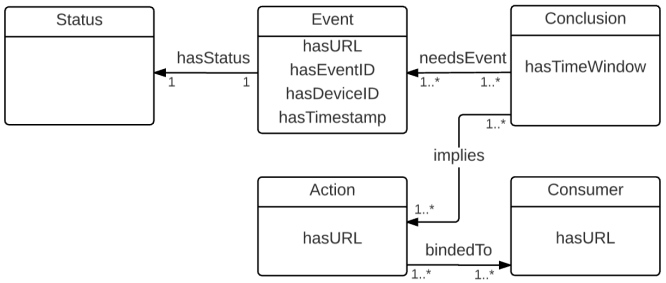

Fig. 3. SECA ontology class diagram

Previous diagram shows the main concepts of the SECA ontology. As it can be seen, the main classes of the model are: *Status*, *Event*, *Conclusion*, *Action* and *Consumer*.

The *Status* concept includes two instances to determine the status of events: *triggered* and *nontriggered*. The *Event* class includes device notified events, which will include their URL, timestamp, device id and event id as *hasUrl*, *hasTimestamp*, *hasDeviceID* and *hasEventID* datatype property values, respectively. In addition, *Event* instances are linked to *Status* instances and updated each time that an event occurrence is notified within the *hasStatus* object property. It should be noted that the initial status of all the described events is *nontriggered*. Afterwards, event status will be updated to *triggered* when notified and its timestamp will be added as a datatype property called *hasTimetamp* with integer data type. In this manner, *Status* class is mainly required at the initial stage because the *hasTimetamp* is updated once the event is triggered.

The *Conclusion* class includes instances that are linked to one or more events. As it is advanced in previous subsection, the approach determines that a conclusion is found when a set of events related to the conclusion occur during the conclusion's time window. Such time window is defined as a *hasTimeWindow* datatype property integer value. Moreover, *Conclusion* instances are linked to *Action* instances, which are actions to be performed once a conclusion is determined. Such link is described within *implies* object property. It should be noted that actions are meant to be service operations that will be invoked within its URL, which is included as a datatype property in *Action* instances.

Finally, Fig. 3 shows the inclusion of *Consumer* class, which will describe the consumers that are mapped to their interested actions. Such relation between actions and consumers is described as *binddedTo* object property. It should be noted that the consumers can be any type of entity that needs to receive any information regarding actions to perform, based on occurrence of related events.

C. Architecture

The objective of this research is to provide a simple and easily reconfigurable approach which processes supply chain events. The simplicity appears in the architecture of the approach. As shown in previous Fig. 1, the KB engine can be seen as a component formed by two main components: an interface (represented as a thin envelop) for communicating with the outside world and the ontology, which describes the required knowledge as semantic descriptions. Following Fig 4 represents the architecture of proposed solution and how it interacts with both providers and consumers of supply chain events.

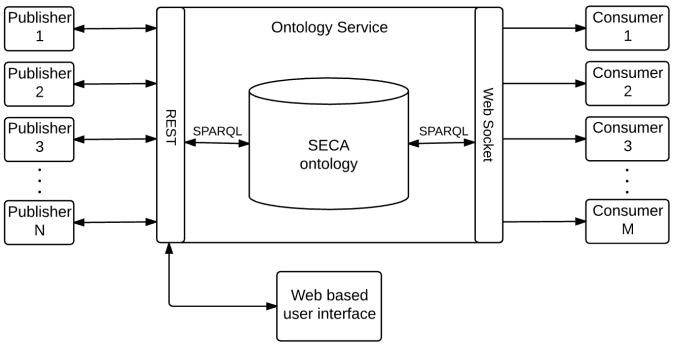

Fig. 4. Proposed architecture

The interface allows the knowledge based engine to interact with other components (user, publishers and consumers). The event processing mechanism is produced inside the SPARQL queries that the interface requests. As Fig. 4 presents, the interface may include different protocols for communications to provide more genericity to the engine.

The deployment of the approach starts with a configuration phase, which consists of designing and uploading the ontology model to the engine. This is done by a user within a web-based user interface. The population of the ontological model can be done within an ontology editor or within SPARQL Update queries because the ontology service can process RDF-based queries.

In addition, for the process to start, the engine must subscribe to all events that it has to handle. Then, the user starts the engine by invoking a service of the interface, which automatically requests all the subscription URLs from the knowledge engine. The SPARQL query shown in Fig. 5 is used to request the events of the system that the interface needs to subscribe.

```
PREFIX seca: http://www.tut.fi/FAST-SECAOnt#
SELECT ?events ?eventURL
WHERE
{
?Conclusion seca:needsEvent ?events.
?events seca:hasURL ?eventURL.
}
```

Fig. 5. SPARQL query used for subscribing to events

In order to describe the interaction of the different components, Fig. 6 presents a sequence diagram. It should be noted that first steps described to be done in configuration phase. This phase ends when the interface sends subscription requests to publishers according to the query result.

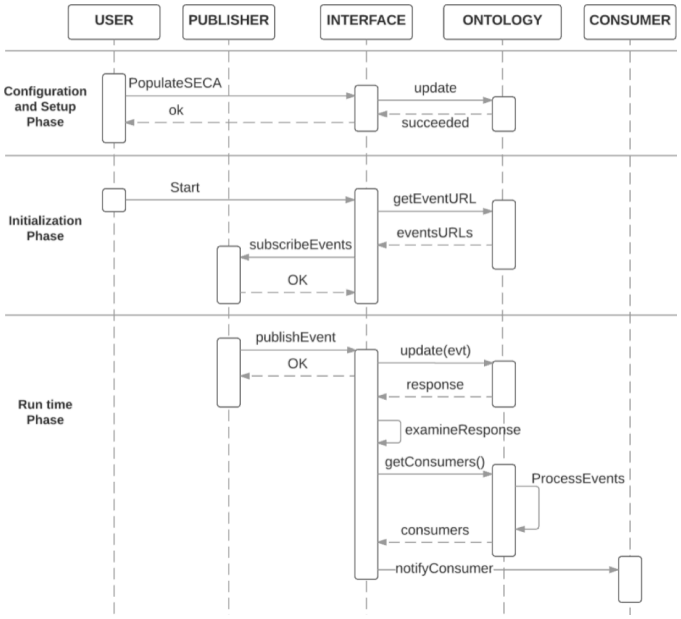

Fig. 6. Sequence diagram for the execution of the event processing

Once an event is triggered by a publisher, the interface updates the event instance information in the knowledge engine. Therefore, the updating operation affects the *hasTimestamp* and the *hasStatus* datatypes values of the event in the ontology model. This update is done within an SPARUL query shown in Fig. 7.

```
PREFIX seca: http://www.tut.fi/FAST-SECAOnt#
PREFIX xsd: http://www.w3.org/2001/XMLSchema#
DELETE{
seca:event_A seca:hasStatus ?oldStatus.
}
INSERT{
seca:event_A seca:hasStatus seca:Triggered.
seca:event_A seca:hasTimestamp "1461920085584"^^xsd:int .
}
WHERE{
seca:event_A seca:hasStatus ?oldStatus.
}
```

Fig. 7. An example for updating the event status after notification

Afterwards, the interface waits until it receives the response from the engine. A simple examination is occurred on the response to assure that the update operation is correctly accomplished. Accordingly, the interface requests list of consumers which can be notified after the update in the event.

In this stage the high-level event processing is achieved. In this research work, the processing procedure occurs inside the SPARQL query. The query depicted in Fig 8 shows the events processing procedure.

```
PREFIX seca: <http://www.tut.fi/FAST-SECAOnt#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?Conclusion ?Action ?Consumer
WHERE
{
{
   SELECT ?Conclusion ?TimeWindowConclusion ?Action
   (MAX(?TimestampEvent) AS ?MaxTS)
   (MIN(?TimestampEvent) AS ?MinTS)
   WHERE {
           ?Event seca:hasTimestamp ?TimestampEvent.
           ?Event seca:hasStatus seca:Triggered.
           ?Conclusion seca:needsEvent ?Event.
           ?Conclusion seca:hasTimeWindow ?TimeWindowConclusion.
       }
       GROUP BY ?Conclusion ?TimeWindowConclusion ?Action
   }
   ?Conclusion seca:implies ?Action.
   ?Action seca:bindedTo ?Consumer
   FILTER(?TimeWindowConclusion >= (?MaxTS - ?MinTS))
}
```

Fig. 8. SPARQL query for processing events

As it can be seen in previous query, the SPARQL query used for processing queries includes a SELECT subquery. The main SELECT clause returns the conclusion, action and consumers by filtering the result according to the time window for each conclusion. Nevertheless, the subquery is needed for returning the upper and lower boundaries of the event for each conclusion. The reason of nesting the subquery is the order of execution in SPARQL. Since we need the maximum and minimum timestamp calculations and SELECT clause is the last one to be executed, the arithmetic operations must be done before last FILTER operator is validated. Once the query is executed, the interface notifies each consumer according to the result of the query.

In this approach, below set of expressions represent the validation when executing the query shown in Fig. 8.

$$t_{max} = max(\forall\, e \in C) \quad (1)$$

$$t_{min} = min(\forall\, e \in C) \quad (2)$$

$$S_c = \begin{cases} YES, timeWindow_c \geq t_{max} - t_{min} \\ NO, timeWindow_c < t_{max} - t_{min} \end{cases} \quad (3)$$

Where, C: conclusion instances in SECA, e: events, t_{max} and t_{min} are the boundaries for the received set of events and S_c states if the conclusion is satisfied, which is expressed as YES/NO value. This operation occurs if and only if all set of events which are needed by the conclusion *hasStatus triggered*.

Finally, the interface receives conclusions with corresponding actions and consumers. Actions could be services invocation from notification or operation in the consumer which provides URL for reachability in the ontology. In this research, the focus is directed towards events processing. Therefore, actions and consumers are limited to notification operations. However, the ontology can be further extended or even merged with other ontologies that might include already such types of descriptions.

IV. Proof of Concept

Once the approach has been described, this section presents a testing case for proving the concept. Firstly, the SECA ontology is populated with several instances to test when a conclusion can be achieved and, consequently, which action and consumer must be executed and advised, respectively. The main objective of this proof of concept is not only to check that mappings between *Status*, *Event*, *Conclusion* and *Action* class instances is correct; but also to test critical timing on triggering of events, which happen in different time window of corresponding conclusions.

A. Testing case

The first step for testing the approach is to populate the ontology with the events that are handled by devices, which are connected to the KB engine interface. In addition, the conclusions to be verified and their links between events and actions are also included in the model. Finally, the consumers that are interested in certain actions, are inserted in the ontology. It should be noted that the designed model has been implemented in OWL, using Olingvo, which is an ontology editor developed at Tampere University of Technology.

Following tables Table I, Table II and Table III show the status of the SECA ontology instances, before uploading the model to the interface. Meanwhile Table I shows the instances of the model, Table II shows the property assertions of instances and Table III shows the datatype property value assertions, which in configuration stage are only the time windows of conclusions. As it can be deducted from the abstract naming of instances, this is an artificial test. Nevertheless, adding correct URLs, this testing model could be employed in a real scenario.

TABLE I. Testing Model Instances

Class	Instances
Status	triggered, nonTriggered
Event	event_A, event_B, event_C, event_D, event_E, event_F
Conclusion	conclusion_A, conclusion_B, conclusion_C, conclusion_D
Action	action_A, action_B, action_C, action_D
Consumer	consumer_A, consumer_B

TABLE II. Property Assertions of Testing Model Instances

Instance	Object property	Instance
event_A	hasStatus	nonTriggered
event_B		nonTriggered
event_C		nonTriggered
event_D		nonTriggered
event_E		nonTriggered
event_F		nonTriggered
conclusion_A	needsEvent	event_A, event_B, event_C
conclusion_B		event_D, event_E, event_F
conclusion_C		event_B, event_F
conclusion_D		event_B, event_C
conclusion_A	implies	action_A
conclusion_B		action_B
conclusion_C		action_C
conclusion_D		action_D
action_A	bindedTo	consumer_A
action_B		consumer_A
action_C		consumer_B
action_D		consumer_B

TABLE III. DATATYPE PROPERTY VALUE ASSERTIONS

Instance	Datatype property	Value
conclusion_A	hasTimeWindow	1000
conclusion_B	hasTimeWindow	500

[a] These values are integers. Then, as an example, 1000 is added in the model as "1000"^^<http://www.w3.org/2001/XMLSchema#integer>

It should be noted that Table II shows the initial status of events as *nonTriggered*. It will be the first event occurrence that will cause the change of status to triggered due to the execution of the query shown in Fig. 7. In fact, Table III does not show any event timestamp because the hasTimestamp value will be updated after the execution of the query shown in Fig. 7. Also, URLs of instances have not been added to Table III because the main focus of this experiment is to verify that conclusions can be correctly mapped to a concrete set of event occurrence. In any case, the URLs are added as *hasURL* datatype property values, which has String type.

B. Results

To proof the concept and, hence, the described approach the experiment has two main tests, according to the sequence diagram shown in Fig. 6. Firstly, the query shown of Fig. 7 must be executed each time that events occur. Then, whenever a conclusion is satisfied by the occurrence of all needed events, the query shown in Fig. 8 must send as an output the mapping between *Conclusion*, *Action* and *Consumer* instances.

In order to depict the times in which certain events have been triggered and, then, the event status of the model updated, Fig. 9 shows a time line of event execution. It should be noted that the unit of time in such diagram is milliseconds.

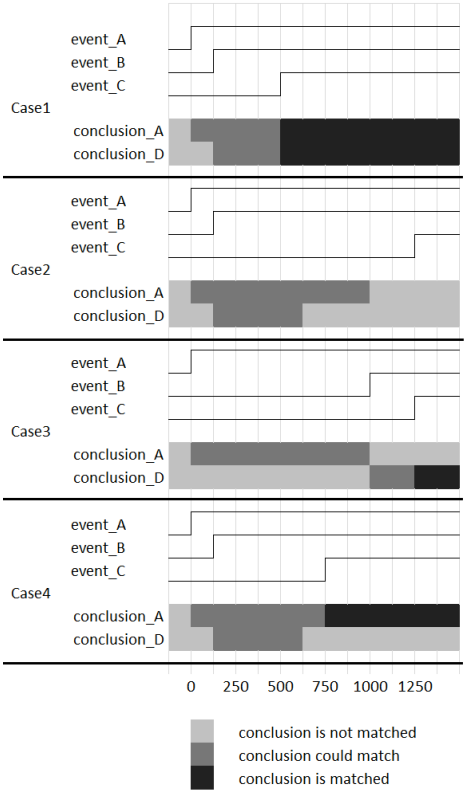

Fig. 9. Timing diagram of event execution

As it can be seen, up to four possible cases between two conclusions that share events has been tested. The graph shows that conclusions are satisfied whenever related events occurred inside their corresponding time window. Finally, to support the proof of concept, Fig. 10 depicts the third case in which *conclusion_A* is not satisfied because even though all needed events have occurred, it has not happened inside the conclusion's time window. Nevertheless, the time window of *conclusion_D* is 500 ms and because of needed events have occurred at timestamp 1000 ms and 1100 ms (they have occurred in 100 ms) they satisfy the condition. Therefore, the result is the satisfaction of *conclusion_D* and, then, the query result also output the mapping between *conclusion_D*, *action_D* and *consumer_B*.

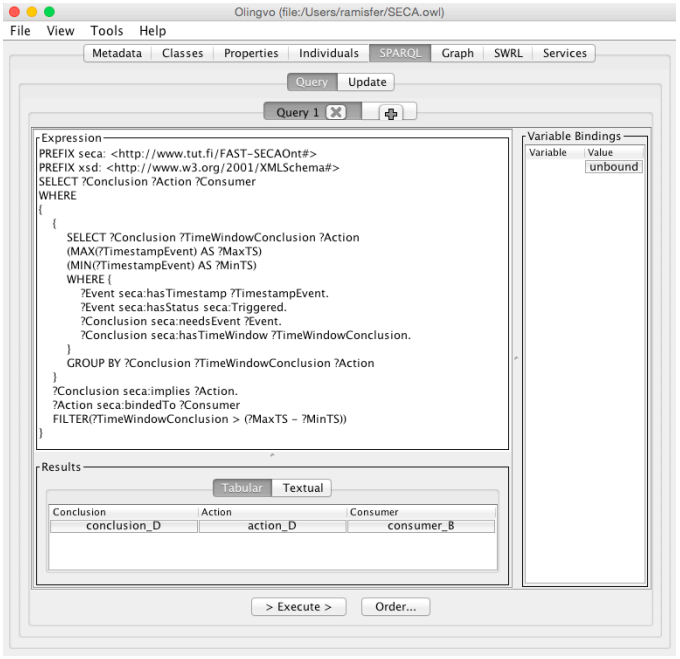

Fig. 10. Query execution and case 3 result validation

V. CONCLUSION

This article presents the concepts, architecture, ontology design and proof of concept for implementing and testing a knowledge-based approach for processing events. The ontology model is hosted by an ontology service that interfaces WS enabled devices and Consumers of event information.

As described, the resulting solution of described research work that might be used to process supply chain events. In principle, such task is normally addressed by CEP tools. The authors of this research claim that presented work is a light alternative to such solutions, which definitely fits in the C2NET framework and can be easily integrated with current modules. In addition, the level of abstraction offered by ontologies, may facilitate the design phase of *Conclusion* instances, which, indeed, would be implemented as rules in CEP engines. In fact, presented solution might be directly deployed in the C2NET platform as a module that interfaces WS enabled devices and systems pertaining or with access to the platform.

Moreover, in terms of configuration, one of the direct benefits of presented approach is the reduction of time and effort that users could experience using CEP approaches based on CEDR queries [14]. Fundamentally, our research work allows users only to take care about definition of relations between events and actions with no need of understanding the insights of the implementation. This is because the queries of the solution are generic and includes the required logic for triggering actions according to their corresponding set of events.

Further, the solution will be tested in a real case scenario, in which the ontology will be populated with real data and service descriptions. Moreover, the speed of communication and amount of events to be handled must be tested before deploying presented solution in the real platform. These tests might imply slight modifications in the ontology service. Besides that, this approach can be extended for achieving more genericity. Consequently, the ontology mode could be expanded as well. For instance, the interface could handle more protocols if these protocols are defined properly in the ontology model. Moreover, during this research work, more ideas and implementations have raised such as including priorities for event and adding the correct definition for the order of received event. In any case, presented research work is the first step for implementing improvements and testing new ideas.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement n° 636909, correspondent to the project shortly entitled C2NET, Cloud Collaborative Manufacturing Networks.

REFERENCES

[1] B. R. Ferrer, S. Iarovyi, A. Lobov, and J. L. M. Lastra, "Towards processing and reasoning streams of events in knowledge-driven manufacturing execution systems," in *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, 2015, pp. 1075–1080.

[2] L. E. G. Moctezuma, J. Jokinen, C. Postelnicu, and J. L. M. Lastra, "Retrofitting a factory automation system to address market needs and societal changes," in *IEEE 10th International Conference on Industrial Informatics*, 2012, pp. 413–418.

[3] M. J. A. G. Izaguirre, A. Lobov, and J. L. M. Lastra, "OPC-UA and DPWS interoperability for factory floor monitoring using complex event processing," in *2011 9th IEEE International Conference on Industrial Informatics*, 2011, pp. 205–211.

[4] I. M. Delamer and J. L. M. Lastra, "Service-Oriented Architecture for Distributed Publish/Subscribe Middleware in Electronics Production," *IEEE Trans. Ind. Inform.*, vol. 2, no. 4, pp. 281–294, Nov. 2006.

[5] Y. C. Dufort, "ISA-95-Based Operations and KPI Metrics assessment and Analysis," White paper 24, A Mesa International, ISA and Ivensys Wonderware co-branded white paper, Nov. 2006.

[6] E. A. Lee, "Cyber Physical Systems: Design Challenges," in *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, 2008, pp. 363–369.

[7] J. Lee, E. Lapira, B. Bagheri, and H. Kao, "Recent advances and trends in predictive manufacturing systems in big data environment," *Manuf. Lett.*, vol. 1, no. 1, pp. 38–41, 2013.

[8] Europäische Kommission, Ed., *Helping firms grow: Commission staff working document SWD(2014)277 final ; [a Europe 2020 initiative]*. Luxembourg: Publ. Off. of the Europ. Union, 2014.

[9] B. Andres, R. Sanchis, and R. Poler, "A Cloud Platform to support Collaboration in Supply Networks," *Int. J. Prod. Manag. Eng.*, vol. 4, no. 1, p. 5, Jan. 2016.

[10] J. Delsing, F. Rosenqvist, O. Carlsson, A. W. Colombo, and T. Bangemann, "Migration of industrial process control systems into service oriented architecture," in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, 2012, pp. 5786–5792.

[11] P. Maheshwari and S. S. Tam, "Events-Based Exception Handling in Supply Chain Management using Web Services," in *Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services (AICT-ICIW'06)*, 2006, pp. 151–151.

[12] T. Lu, X. Guo, B. Xu, L. Zhao, Y. Peng, and H. Yang, "Next Big Thing in Big Data: The Security of the ICT Supply Chain," in *2013 International Conference on Social Computing (SocialCom)*, 2013, pp. 1066–1073.

[13] Baofeng Huo, Yinan Qi, Zhiqiang Wang, and Xiande Zhao, "The impact of supply chain integration on firm performance: The moderating role of competitive strategy," *Supply Chain Manag. Int. J.*, vol. 19, no. 4, pp. 369–384, Jun. 2014.

[14] P. Pietrzak, P. Lindgren, and H. Mäkitaavola, "Towards a lightweight CEP engine for embedded systems," in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, 2012, pp. 5805–5810.

[15] B. Ramis, L. Gonzalez, S. Iarovyi, A. Lobov, J. L. M. Lastra, V. Vyatkin, and W. Dai, "Knowledge-based web service integration for industrial automation," in *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, 2014, pp. 733–739.

[16] S. Iarovyi, W. M. Mohammed, A. Lobov, B. R. Ferrer, and J. L. M. Lastra, "Cyber-Physical Systems for Open-Knowledge-Driven Manufacturing Execution Systems," *Proc. IEEE*, vol. 104, no. 5, pp. 1142–1154, May 2016.

[17] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing?," *Int. J. Hum.-Comput. Stud.*, vol. 43, no. 5, pp. 907–928, 1995.

[18] D. Kalibatiene and O. Vasilecas, "Survey on Ontology Languages," in *Perspectives in Business Informatics Research*, J. Grabis and M. Kirikova, Eds. Springer Berlin Heidelberg, 2011, pp. 124–141.

[19] "OWL Web Ontology Language Reference." [Online]. Available: https://www.w3.org/TR/owl-ref/. [Accessed: 01-May-2016].

[20] "RDF - Semantic Web Standards." [Online]. Available: https://www.w3.org/RDF/. [Accessed: 01-May-2016].

[21] "SPARQL Query Language for RDF." [Online]. Available: https://www.w3.org/TR/rdf-sparql-query/. [Accessed: 01-May-2016].

[22] "SPARQL 1.1 Update." [Online]. Available: https://www.w3.org/TR/sparql11-update/. [Accessed: 01-May-2016].

[23] "SWRL: A Semantic Web Rule Language Combining OWL and RuleML." [Online]. Available: https://www.w3.org/Submission/SWRL/. [Accessed: 01-May-2016].