Sensor SelComp, a Smart Component for the Industrial Sensor Cloud of the future

Luis Neto, João Reis, Ricardo Silva, Gil Gonçalves

{lcneto, jpcreis, rps, gil}@fe.up.pt
FEUP, Faculdade de Engenharia, Universidade do Porto, Porto, Portugal
ISR-P, Instituto de Sistemas e Robótica - Porto, Portugal

***Abstract*—Industrial Internet of Thing's will pave the way for Smart Manufacturing initiatives. Intelligence will be notices even in fine grained devices, resulting in complex but at the same time highly efficient manufacturing systems. The typical field worker job will be replaced by machines and human intervention will be shifted to supervision jobs. This work presents Sensor SelComp, a Smart Component which will act in the factory shop-floor, creating the so called digital twin's of machine's by means of sensors and exposing it's functionalities as services. This component is a building block of the SelSus project vision, whose aim is to create and highly effective self-healing production environment. This component eases the process of sensor integration and data analysis by offering runtime reconfiguration and data processing capabilities. Along this document it is shown that Sensor SelComp can cope with tight industrial functional requirements and it's functionalities are described in detail.**

Keywords–Smart Component; Industrial Sensor Cloud (ISC); Industrial Wireless Sensor Network (IWSN); Software Reconfiguration; Industrial Cyber Physical Systems(ICPS); Industrial Internet of Thing's (IIoT);

I. Introduction

The work presented in this paper is part of SelSus European Project, whose objective is the development of a diagnostic and prognosis environment [1], aware of the condition and history of the machine components within a system or factory for highly effective, self-healing production resources and systems to maximize their performance over longer life times through highly targeted and timely repair, renovation and upgrading. Exploring Wireless Sensor Networks (WSNs) and Cloud Systems in Industry that ranges from sensor integration, sensor data visualization, statistical processing and access, where sensors, external to the process used for machine monitoring were introduced at the shop-floor level, enabling self-aware and self-healing production systems. This work focus on a central piece of this architecture, the Sensor SelComp [2], a smart component orchestrating WSN under SelSus Industrial Sensor Cloud domain.

The nature of industrial environment, namely, shop floor operation, has shaped the requirements of this solution in a very specific way. There are some abstract concepts among the IIoT (Industrial Internet of Thing's), that can be applied to industry in general [3]. Such is the case of division in layers (perception, networking, service), adoption of SOA (Service Oriented Architecture) concept's and agreement on standards for M2M (Machine to Machine) communication and service interoperability. But when there is the need to tailor these building blocks into something concrete and applicable in real case scenarios, these techniques lacks detail and a lot of unpredicted peculiarities interfere.

Economical, technological and operational differences makes every smart factory implementation process unique. Recurrent difficulties, such as the lack of common agreement on service description language, leveraged to create a new specification to suite the SelSus project needs. Another fundamental difference is the hardware used. A company can at principle buy sensors from the same manufacturer, but new requirements and the continuous offer of solutions in the market, makes hardware in use to change. Older sensors are maintained and new ones are introduced in the system. The same happens with machines, software, processes, services and so on. The Sensor SelComp solution is designed to cover some of this issues that are inherent to the transient nature of the smart factory environment.

There are three [4] frequent and common challenges in IoT that are addressed in this work:

1) Currently WSN (Wireless Sensor Networks) are composed of heterogeneous sensor nodes. This constraint force developers to know low level details about protocols used by the sensors they are trying to integrate.
2) There is also a semantic gap between the proprietary representation of low level data and the high level needed to develop applications on top of the sensors.
3) Because of previous gap's, a great effort is also required to build high level applications. There is a need to transform the control and functionalities of sensor nodes in services for the high end developer to use.

The rest of the paper is organized as follows: Section II, dives into the Smart Component philosophy; Section III, describes intrinsic characteristic of the Sensor SelComp implementation; Section IV, reports some of the plugin's built for Sensor SelComp and tests made; Section V, results from the previous section are discussed; Section VI, some conclusions are drawn and future work is presented.

II. Sensor SelComp

The Sensor SelComp technology inherits its main characteristics from the Smart Component philosophy. This philosophy is based in a consistent study of the Smart Manufacturing

initiative and it is being systematically refined and matured by past and present European projects (*XPress, IRamp3 and SelSus*). There are five essential characteristics to a Smart Component:

- **Reconfigurable and modular**; the solution must be capable of extend its capabilities by adding new software modules and it must be capable of reconfigure it's internal operation in runtime.
- **Data processing capabilities**; system state assessment, event detection and fault alarm requires data processing capabilities.
- **Omnidirectional communication and interface capabilities**; omnidirectional means that the system must be capable to talk with devices at a lower level (sensors and machines), same level (other Smart Component's) and higher level (cloud servers, manufacturing systems).
- **Process events and take actions**; this capability provides the system with a certain degree of smartness and autonomy. In case any event of interest, the system must be capable of detecting it and take the proper actions.
- **Real-time acquisition, processing and delivering**; typically, field devices operate at variable real-time scales, performing multiple tasks in a coordinated way. Provide actions in real time is a vital factor for industrial scenarios.

Due to previous characteristics, such a system must adopt the ontology from the environment that surrounds it. This characteristic makes the Smart Component fully adaptable and ready to integrate in any manufacturing system. The ontology of the SelSus System, it is characteristic, that in practice, converts a Smart Component into a Sensor SelComp. By adopting the ontology defined in the project (*SelComp Self Description* and *Data Payload*), SelComp is capable to interact with any actor within the SelSus environment.

It is also necessary to urge the increasingly importance to develop components that are embeddable into advanced production environments. Such is the case of Industrial Cloud Computing (ICC) and Industrial Cyber Physical Systems (ICPS). The proposed solution was designed with this necessity in mind, some benefits of such approach are pointed next. Some advantages of support such technologies summarized in [5] are: 1) physical hardware and software used in the factory can be intelligently perceived, therefore, the information collected can be correctly processed and used in the whole life-cycle of production. 2) Cloud Computing applications can work over large quantities of data enabling production-related & product-related services [6] and even more advanced business models [7]; 3) smart logistics of both machine and human labour as well the energy spent.

The Smart Component is able to span vertically over the four-layered architecture proposed [3] for IIoT, at the same time, cooperating horizontally at each layer of abstraction (Fig. 1).

Sensor SelComp transforms heterogeneous devices in compatible services, any application or device in the network can use any possible combination of services. In [8] a middleware component, *DREAMS*, features some interesting

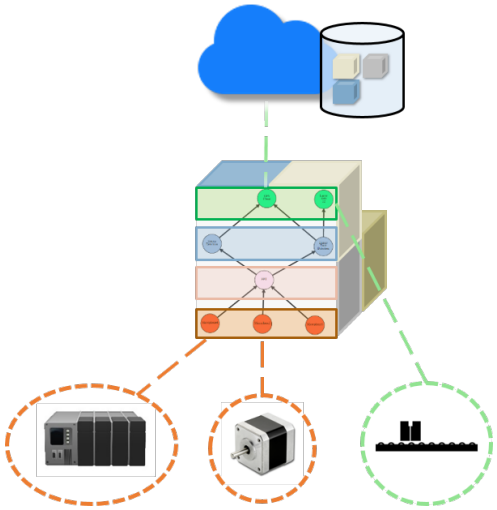

Figure 1: Sensor SelComp layers of interaction.

characteristics. The component have a *Module Pool* with a clever organization of modules, identified by characteristics such as the amount of memory and energy they consume. A reconfigurable smart sensor module is presented in [9]. This component implements the IEEE1451.2 [1] standard. It defines a smart transducer with reconfiguration capabilities to interface sensors and actuators.

III. Implementation

A. SelSus System Interfacing

In a traditional Service Oriented Architecture (SOA) there are three fundamental parties [10]: service provider, consumer and broker. In the SelSus system, every node is both a consumer, provider and broker, the system itself was designed to run free of fixed service registry. In the chosen approach, each node must describe its services and ask for other's service descriptions. This mesh style approach to SOA requires that every node share the same ontology, independently of the kind of hardware that is being abstracted. Beyond a common service description, connectivity requirements of ICPS [11] (real-time data acquisition and feedback, intelligent data management, *Plug& Play*), leveraged the adoption of a common data payload format. The advantages of such are mostly reflected in

[1] IEEE1451 standard: http://ieee1451.nist.gov/senoct11-2col.pdf

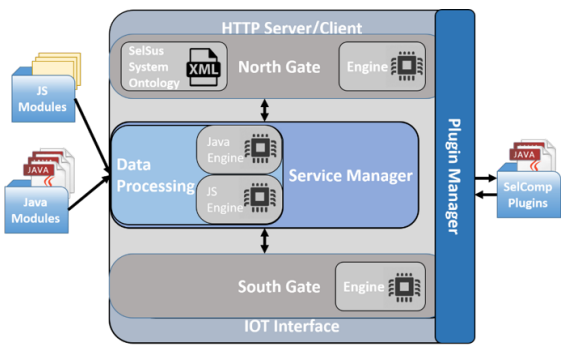

Figure 2: Sensor SelComp architecture.

Plug & Play and consequently in the Sensor SelComp and SelSus System reconfiguration.

B. Internal Structure

A Sensor SelComp can receive sensor data from distinct physical devices and application protocols. That data can be treated by different algorithms, which are encapsulated in modules. Once instantiated, these act as services which subscribe and publish data. Any active instance of sensors, data treatment modules and SelSus System devices (Sensor Cloud, other SelComp's and HMI device's) can be combined in a way that can be represented as a graph (Fig. 3).

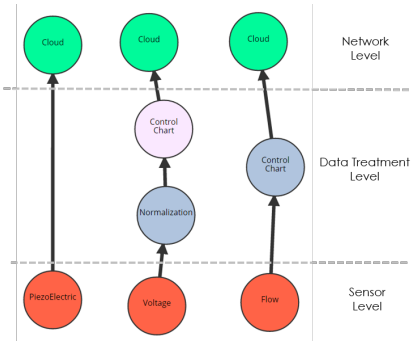

Figure 3: Internal Sensor SelComp configuration.

The Sensor SelComp internal logic arrangement, is represented using a *directed acyclic graph (DAG)*. The graph structure in Fig. 3, which represents the internal flow of information within Sensor SelComp. It can be divided into three levels, each with a different label and colour assignment: the Sensor Level includes sensor instances (bottom level), providing data to the gateway; the data treatment level (middle level), includes nodes representing instances of algorithms embedded at the gateway that can treat information in several ways (eg: aggregate data using mean or other functions, perform trend analysis); the Network Level (top level) includes nodes where the flow resulting from the lower level nodes can be redirected to subscribing hosts in the network. Each edge of the graph has a buffer that can be adjusted in size. When a certain buffer reaches its size, the whole content will be sent to the subscribing nodes. Adjustment in buffer size allows to select the number of samples and also to synchronize time differences between sensors; which can be useful if data from two sensors with discrepant sampling frequencies is needed at the same time. This internal structure can be dynamically rearranged in runtime: new sensors and data modules can be loaded into the Smart Node; the connections between nodes can be reformulated to synchronize and treat data in new ways.

C. Dynamic Modular Software Reconfiguration (DMSR)

The Dynamic Modular Software Reconfiguration is the proposed solution embedded in the Sensor SelComp to be highly reconfigurable and to quickly adapt industrial demands. Therefore, it must provide consistent mechanisms to configure, deploy and dynamically reconfigure the Sensor SelComp source code responsible for data processing, which traduces in a component-based middleware. It is component-based because each piece of software that can be reconfigured at the Sensor SelComp level is seen as a component that can be easily updated or exchanged.

In software engineering the terms Modular Software and Software Component are the most approximate topics to what we refer as Reconfigurable Software. Component reuse, reduction of the production cost, reconfiguration in runtime, short time to market and systematic approach to the system construction are some of the key benefits of using a component model. Component models can be divided in two categories: 1) as in object-oriented programming, components are objects; 2) components represent units in software architectures. A generally accepted view of a software component is that it is a software unit with provided services and required services." [12]

In the proposed solution two component model approaches are offered to provide an even better flexibility. At the system level there is the possibility to extend the Sensor SelComp building blocks with plugins. At data processing level, there is the possibility to add new data treatment and processing modules to the Sensor SelComp to use. There are two ways to develop data treatment modules: 1) develop code in Java using the defined interfaces; 2) develop code in C#, Java or C++ and use the SelSus Cloud to convert the code in a JavaScript module file. In both ways, the SelSus Sensor Cloud uploads the modules to the Sensor SelComp, which maintains the modules in the local file-system Figure 2.

A SelComp internal logic arrangement is represented using a directed acyclic graph (DAG). The graph structure in Figure 3 can be divided in three levels, each with a different label and color assignment: the Sensor Level includes sensor instances (bottom level), providing data to the gateway; the data treatment level (middle level), includes nodes representing instances of algorithms embedded at the gateway that can treat information in several ways (e.g. aggregate and validate data using techniques such as control charts, perform trend analysis, etc.); the Network Level (top level) includes nodes where the flow resulting from the lower level nodes can be redirected to subscribing hosts in the network (e.g. Sensor Cloud, industrial machines). This internal structure can be dynamically rearranged in runtime using the Sensor Cloud, where new sensors and data modules can be loaded and therefore, the connections between nodes can be reformulated to synchronize and treat data in new ways.

1) Data Processing Modules: To increase the system flexibility, the *Data Processing* unit was designed to run *Java* and *JS* modules. The *Java* approach still mandatory when exists the necessity to use system level *API's* and higher data processing speeds. Nevertheless, the *Rhino JS Engine*[2] engine used has the capability to load *JAR API's*. The life-cycle requires different approaches in both cases:

- *JS* modules: These modules can be developed directly in *JS* or using *C, C++, C#* or *Java*. The developer has to implement a prototype function which will be invoked in runtime. The JSIL compiler [13], installed in the Sensor Cloud, converts the code in *JS* and uploads it to the chosen Sensor SelComp. Once the *Nort Gate* detects a new module, it is stored in the local file-system and a new *Rhino* engine instance compiles the new file.

[2]https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino

- *Java* modules: Must implement the data processing abstract interface and be identified with *Name, Version* and *Type String* fields. The modules are also sent to the desired SelComp by the Sensor Cloud. Via *reflection*, *.class* files are catalogued using the three key fields so later they can be instantiated using the same strategy.

In both chases there are two validation steps that guarantee the correct type when interfacing any two modules. The first step is performed in the Sensor Cloud; the user must prompt some information about the modules and the input and output types as well. The Sensor Cloud does not issue reconfiguration processes unless all input and output types of connected modules are compatible.

2) Sensor SelComp Plugins: The constant appearing of new IoT hardware solutions and the lack of standards to integrate them justified an extra effort to make the SelComp software capabilities extensible. Sensor Cloud hosts and maintains the plugins in a central repository. In the SelComp, plugins are maintained in the local file-system, the deployment and *life-cycle* processes of a certain plugin are controlled by Sensor Cloud. All plugin implementations provide *start, stop* and *restart* functions. In this context, a plugin can be a piece of software to drive hardware and also purely software components. The diagram of Figure 4 shows the present plugins that can be used to acquire and send sensor data. The blocks in grey represent plugins that are being developed to extend the functionalities of Smart Component.

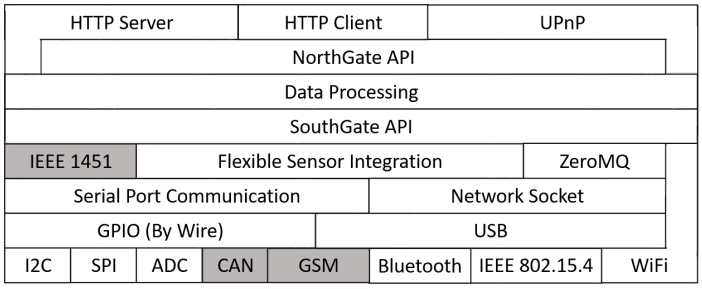

Figure 4: Sensor SelComp Stack.

D. Sensor Integration

The aim of the *South Gate* module was to be as most expandable as possible. By expandable, it is intended that the component should be able to work at least at three levels of abstraction: 1) to embed new communication protocols; 2) to incorporate proprietary driver software, wrapped by the plugin interface; 3) to directly connect sensors by wire, using the embedded platform libraries. In addition, a *flexible sensor integration* methodology was developed as a plugin. This component listens to all free serial ports to find a data input. Once a data source is found, it buffers the input and tries to parse it using a list of available *XML* files in the local file-system. These *XML* files describe the fields of a data packet.

IV. Testing

The modules developed for sensor integration described in this section can be better perceived by analysing Figure 4. The blocks filled in grey are plugins that will be developed as future work. All the other blocks form a flexible solution, which can cope with the majority of the scenarios in which IIoT plays a fundamental role.

This section describes briefly the different blocks implemented and relies on design considerations for industrial IoT applications (*"From the technology perspective, the design of an IoT architecture needs to consider extensibility, scalability, modularity, and interoperability among heterogeneous devices."*) [3] to provide the tests needed to proof the Smart Component robustness. The testing process was split according with two major considerations. One off the major issues was to verify if the solution was able to scale, using a reasonable number of nodes (Subsection IV-2). Due to the time scale requirements in industry operation, the solution was tested to a reasonable frequency of operation (Subsection IV-5).

1) Spectrometer: One of the issues of using a *Java* approach, is the abstraction level, which can be both a barrier considering low level at which sensor driver solutions are implemented and the loss in performance. To mitigate these issues a solution based in *IEEE1451* is being designed. In the current state of development, a provisional solution was adopted. Using *Java Native Interface (JNI)*, a solution was developed to communicate with a *Dynamic Link Library (DLL)* from the spectrometer sensor manufacturer. The SelComp was able to fetch and treat 54000 data points sampling per 1Hz.

2) Motes: Motes are programmable boards which embeds a set of sensors, typically have Radio Frequency (RF) communication capabilities and there are proprietary and open source solutions. Solutions for both proprietary (Libelium Waspmote V1.2 [3]) and open source (MTM-CM5000 (TelosB generic) [4]) were developed and tested. The proprietary drivers and software libraries are used to build up plugins to integrate these devices. Such is the case of *Tmote*, *Libelium*, *Mica2*. All of the above mentioned devices were integrated in such a way that they can have a variable number of sensors. These boards provide a fixed number of sensors, anyway, the number of sensors in the board can be increased or reduced. This changes cause the messages being sent to vary in size. The plugins developed have the capability of report values from all the sensors, even in case they change in number.

TABLE I: SCALABILITY RESULTS

	Buffer Size			
	1			
Sensor Nodes	10		10	
End Nodes	10		10	
Data Treatment Nodes	0		10	
	Sent/Received	Sent	Sent/Received	Sent
HTTP Packets (24 hours)	2914694	1338592	1304250	633506
Avg. Packets sec	33.7	15.5	15.1	7.3

For scalability testing, a scenario featuring 10 MTM-CM5000 motes was set. The Sensor SelComp was configured to receive data of two sensors (Invisible and Visible Light), from the 10 motes, sampling at 2Hz each. Two tests were performed: in the first one, the sensor data was sent in raw format, representing 20 nodes in the internal structure; in the second one, the flow of data from each sensor was treated

[3] Waspmote V1.2 datasheet: http://www.libelium.com/downloads/documentation/v12/waspmote_datasheet.pdf

[4] MTM-CM5000 features: http://www.advanticsys.com/shop/mtmcm5000msp-p-14.html

using a data normalization JS module, representing 30 nodes in the internal structure.

3) UPnP: As a stack of protocols, *UPnP* (Universal Plug&Play) offers a set of advantages that regards a lot of the necessities exposed throughout the document. The necessity of interconnect devices, which can share services with the minimum effort from the users to do configurations has been the principal motivation behind this architecture protocol. This architecture was introduced by *Microsoft* with a "connected home" aim, currently is maintained by the *Open Connectivity Foundation*[5]. This protocol is independent from the physical platform, supports *zero configuration* in lack of *DNS* servers and promotes device connectivity allowing devices to discover each one services, subscription of events and control. An UPnP plugin was developed to benefit from: 1) Sensor Cloud communication alternative; 2) sensor interfacing; 3) inter-SelComp communication.

4) ZeroMQ: Modern messaging protocols are becoming popular [14] options for IoT. Its lightweight implementations and multiple architectural possibilities make them suitable for a variety of applications, such as industrial internet solutions [15]. For these reasons, and also to test sensor data acquisition in the SelSus Project using this technology, a *ZeroMQ* plugin was developed. The *Dealer to Dealer Request-Reply combination*[6] was implemented.

5) Serial Port Interface: Given the requirements of solutions oriented to industrial application, the best way to acquire real time data is by wiring sensors directly to the Smart Component. The protocols used are shown below the GPIO (general purpose input/output) block in Figure 4. To perform tests, a thermocouple sensor was attached by wire to the Smart Component. Using the Serial Peripheral Interface (SPI) plugin, the sensor was integrated; the plugin also provides acquisition frequency control. By increasing the sampling frequency, the throughput of the Smart Component was measured, with and without data treatment modules in the data flow. The wired sensor interfacing was chosen to test the solution in terms of throughput because is the most suitable way to test with high input acquisition rates. The tests were performed in a *Raspberry Pi 2* platform, running *Raspbian Linux* and using a single *MAX31855* thermocouple sensor. The complete SelComp experiment set-up was connected to the Sensor Cloud through *Ethernet* cable.

Essentially two metrics was used to evaluate the reliability of the solution, throughput capacity (Table II) and reconfiguration speed (Table III). There are three variables that have the most impact in the results obtained: 1) the number of data treatment nodes in a data flow of a sensor; 2) the acquisition frequency of the sensor; 3) the sizes of buffers in each edge of the graph (Figure 3).

Each individual test reported in Table II was run for 60 minutes. For each of the tested frequencies, the SelComp was tested with data treatment nodes during at least 24 hours to ensure that in a continuous processing the solution would not fail. As reported in subsection III-C1, it is possible to run data processing modules written in Java or JS. The modules used in the tests were written in JS, which is the option with heaviest processing requirements. The buffer which varies in the tests, is the one associated to edge between the sensor node and the node after. Varying this buffer allows to establish a trade-off between sending data to cloud at higher rates and send few packets with more data.

[5]UPnP Protocol: https://openconnectivity.org/

[6]ZeroMQ manual: http://zguide.zeromq.org/page:all

TABLE III: RECONFIGURATION RESULTS

Buffer Size	Module Load (s)	Reconfiguration (s)	Sensor Sampling (Hz)
2	16.174	3.432	10
	17.35	3.453	
50	15.984	3.454	
	16.776	10.795	
2	18.252	3.436	50
	16.659	3.509	
50	18.176	3.485	
	17.466	3.306	
2	18.683	12.293	100
	18.104	3.487	
50	18.841	11.962	
	19.901	3.475	
Average (s)	17.69714583	5.50725	

Reconfiguration process, in the worst case, requires two operations. The results in Table III were collected for the worst case, which involves transfer of modules to Sensor SelComp. If any module required by the reconfiguration process is not present locally, Sensor Cloud first transfers the module, and secondly, it sends a reconfiguration message.

V. Discussion

Regarding the results, a loss of throughput when applying processing to data can be perceived in Table II. The number of packets transmitted it is reduced by the double each time a processing node is added to the data flow. The throughput difference can be explained by taking in account that the minimum buffer value for any edge is 2. The reason is that for any processing operation, it is safer to provide at least two values to the function treating data. If there is the need to perform an operation with multiple values, it is always guaranteed that the processing will not fail. This means that if the buffer size was 1, the impact in throughput results would be minimal and the solution would be able to deliver at high rates to Sensor Cloud. In terms of reconfiguration, the speed of operation is shown to slow when sampling data at higher rates. Despite this fact, time results obtained are adequate to the kind of operation being performed, since the reconfiguration process runs smoothly even in high frequency data collection. Regarding the quality and quantity of data, there is a trade-off by varying the buffer size in the sensor edge. In a situation which requires that data reaches the Sensor Cloud at higher rates, a small buffer value provides an adequate response, although it is uncommon to send data to Cloud at those speeds. At small frequencies, the quantity of sensor data is the same, however, total number of bytes transmitted is far less in this case. This can be of the higher importance if we consider a network were multiple Sensor SelComp's, machines and other devices are flooding the network with information. In terms of throughput, it can be verified that the impact caused by the size of the buffer is smaller when the number of nodes increases. If there are no data treatment nodes, the number of packets sent is always at least the double when we use the smaller buffer size. When two data treatment nodes are used, for the same sampling frequency, the number

TABLE II: THROUGHPUT RESULTS

	Sent/Received	Sent	Sent/Received	Sent	Sent/Received	Sent	Sent/Received	Sent	Sent/Received	Sent	Sent/Received	Sent	Data Treatment (nodes)
Sampling (Hz)	10				50				100				
Buffer Size	2		50		2		50		2		50		
HTTP Packets (60 min)	182010	9086	810	381	65998	32977	6511	3244	15041742	6556288	11936	5959	
Avg. Packets sec	5010	2.5	0.22	0.1	18	9	1.8	0.9	84934	42458	3.2	1.6	
Avg. Packet size	733	921	1213	1103	709.5	897.5	1275.5	1113.5	23.1	11.6	1260.5	11000.5	No
Avg. Bytes/sec	3673.08	2301.75	270.77	115.85	12000	8057	2261	983	707.5	893.5	4044	1762	
Bytes	13351457	8366737	982755	420221	46842895	29588715	8304415	3612627	16000	10000	15041742	6556288	
HTTP Packets (60 min)	17794	8876	18376	9136	62067	31004	17774	8857	144407	72184	33837	16901	
Avg. Packets sec	4.8	2.4	2.5	1.2	16.9	8.4	4.5	2.4	39.3	19.6	9.2	4.6	
Avg. Packet size	717.5	905.5	735.5	896.5	1026.5	879.5	694.5	881.5	734.5	922.5	714.5	901.5	1
Avg. Bytes/sec	3476	2188	1829	1108	17000	7424	3147	2126	28000	18000	6578	4148	
Bytes	12767329	8034890	14000	8866	63741894	27257008	12336550	7809054	106121006	66582175	24159950	15235118	
HTTP Packets (60 min)	8646	4319	8319	4137	9959	4951	8902	4381	29705	14839	24525	12045	
Avg. Packets sec	3.1	1.5	2	0.1	2.7	1.3	0.9	0.8	8.1	4	0.4	0.2	
Avg. Packet size	715.5	901.5	710.5	895.5	721.5	909.5	710	893	707.5	909.5	690.5	882.5	2
Avg. Bytes/sec	2199	1389	1446	97	1955	1226	687	561	5884	3675	265	166	
Bytes	12183201	7703043	5909298	3703824	7181540	4502982	4557730	2856788	21608357	13495477	16946440	10634659	

of packets sent is much more similar than when compared with the same frequency without data treatment. In terms of scalability (Table I), the solution shows a linear behaviour in comparison with throughput results. The number of nodes can be considered few, but these results were obtained using real hardware available and running continuously for 24h, which proves the scalability.

VI. Conclusion and Future Work

Additionally to the testing, this solution was also deployed in the demonstrators of the SelSus project and proved that it can offer the it's core functionalities (reconfiguration and easy sensor integration capabilities) under real operation scenarios. One drawback of this solution is that it requires development of new modules to adapt to new situations. But once the plug-ins are developed, they are reusable and this compensates the effort in develop new software. To take advantage of this re-usability, a distributed repository of plug-ins can be created, and this way, the community can share solutions. A priority in future work will be the focusing in industrial protocols and standards for plug-in development. In Figure 4, are identified two plug-ins which represent a concern step into industrial environment. The IEEE 1451 plug-in, for automatic sensor information retrieval, will eliminate much of the effort creating new sensor plug-ins by automatically filling in sensor information and properties. Regarding wired sensor integration, CAN protocol is widely used in industrial environments and will be a priority regarding plug-in development. Considering the end user control interface in Figure 1, as well as the component model developed to create new modules (Section III-C), standards exist [16] and they will be investigated as current H2020 European Projects for Smart Industry [17] initiatives are pointing out in that direction.

Acknowledgment

SelSus EU Project (FoF.NMP.2013-8) Health Monitoring and Life-Long Capability Management for SELf-SUStaining Manufacturing Systems funded by the European Commission under the Seventh Framework Programme for Research and Technological Development.

STRIDE Smart Cyber-physical, Mathematical, Computation and Power Engineering Research for Disruptive Innovation in Production, Mobility, Health, and Ocean Systems and Technologies - funded by program N2020 (2016-2020) supported partially by FEDER.

References

[1] M. S. Sayed, N. Lohse, N. Sondberg-Jeppesen, and A. L. Madsen, "Selsus: Towards a reference architecture for diagnostics and predictive maintenance using smart manufacturing devices," 2015.

[2] L. Neto, J. Reis, D. Guimarães, and G. Gonçalves, "Sensor cloud: Smartcomponent framework for reconfigurable diagnostics in intelligent manufacturing environments," in 2015 IEEE 13th international conference on industrial informatics (INDIN). IEEE, 2015, pp. 1706–1711.

[3] L. Da Xu, W. He, and S. Li, "Internet of things in industries: A survey," IEEE Transactions on Industrial Informatics, vol. 10, no. 4, 2014, pp. 2233–2243.

[4] F. C. Delicato, P. F. Pires, L. Pirmez, and T. Batista, "Wireless sensor networks as a service," in Engineering of Computer Based Systems (ECBS), 2010 17th IEEE International Conference and Workshops on. IEEE, 2010, pp. 410–417.

[5] F. Tao, Y. Cheng, L. Da Xu, L. Zhang, and B. H. Li, "Cciot-cmfg: cloud computing and internet of things-based cloud manufacturing service system," IEEE Transactions on Industrial Informatics, vol. 10, no. 2, 2014, pp. 1435–1442.

[6] G. Allmendinger and R. Lombreglia, "Four strategies for the age of smart services," Harvard business review, vol. 83, no. 10, 2005, p. 131.

[7] D. Wu, D. W. Rosen, L. Wang, and D. Schaefer, "Cloud-based design and manufacturing: A new paradigm in digital manufacturing and design innovation," Computer-Aided Design, vol. 59, 2015, pp. 1–14.

[8] A. El Kouche, L. Al-Awami, and H. Hassanein, "Dynamically reconfigurable energy aware modular software (dreams) architecture for wsns in industrial environments," Procedia Computer Science, vol. 5, 2011, pp. 264–271.

[9] Q. Chi, H. Yan, C. Zhang, Z. Pang, and L. Da Xu, "A reconfigurable smart sensor interface for industrial wsn in iot environment," IEEE Transactions on Industrial Informatics, vol. 10, no. 2, 2014, pp. 1417–1425.

[10] A. Arsanjani, "Service-oriented modeling and architecture," IBM developer works, 2004, pp. 1–15.

[11] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for industry 4.0-based manufacturing systems," Manufacturing Letters, vol. 3, 2015, pp. 18–23.

[12] K.-K. Lau and Z. Wang, "A survey of software component models," in in Software Engineering and Advanced Applications. 2005. 31 st EUROMICRO Conference: IEEE Computer Socity. Citeseer, 2005.

[13] K. Gadd. Jsil compiler. [Online]. Available: http://jsil.org/ (2016)

[14] S. Bandyopadhyay and A. Bhattacharyya, "Lightweight internet protocols for web enablement of sensors using constrained gateway devices," in Computing, Networking and Communications (ICNC), 2013 International Conference on. IEEE, 2013, pp. 334–340.

[15] A. Buda, K. Främling, J. Borgman, M. Madhikermi, S. Mirzaeifar, and S. Kubler, "Data supply chain in industrial internet," in Factory Communication Systems (WFCS), 2015 IEEE World Conference on. IEEE, 2015, pp. 1–7.

[16] V. Vyatkin, "Iec 61499 as enabler of distributed and intelligent automation: State-of-the-art review," IEEE Transactions on Industrial Informatics, vol. 7, no. 4, 2011, pp. 768–781.

[17] D. Rotondi, E. Coscia, K. Fischer, L. Aştefănoaei, M. Rooker, M. Isaja, O. Botti, and S. Voss, "D2. 1-d 2.1 a beincpps architecture and business processes," Update, vol. 1, 2016, p. 13.