Universal Parser for Wireless Sensor Networks in Industrial Cyber Physical Production Systems

Ricardo Silva, João Reis, Luís Neto and Gil Gonçalves
Instituto de Sistemas e Robótica
Faculdade de Engenharia da Universidade do Porto
Rua Dr. Roberto Frias, s/n
4200-465 Porto
PORTUGAL
Email: { rps, jpcreis, lcneto, gil }@fe.up.pt

***Abstract*— This work was developed in the context of European funded Project SelSus. Industrial clouds are heavily sensor based and Cloud Manufacturing Service frameworks are mostly grounded in the adoption of Internet of Things and Wireless Sensor Networks technologies. The SelSus framework combines both an Industrial Sensor Cloud and a Cyber Physical Production System. The Industrial Sensor Cloud supports the Cyber Physical Production System, by providing computational power, as well as internal coordination and control, and external access to realize intelligent monitoring and control. The SelSus framework combines embedded systems, networks, sensors and actuators and control algorithms in a seamless manner. Our goal is to have a Flexible Sensor Integration solution that allows rapid graphical development of interpreters of raw data packets in the Cloud and its deployment for embedded execution at the Wireless Sensor Network gateway level for automatic data acquisition. This paper describes such a technology and demonstrates its feasibility, where a match between a graphically developed interpreter and the received messages from the Wireless Sensor Network in US-ASCII is made, and the integration of such sensors is made into the system.**

***Keywords*—Industrial SelSus Cloud (ISC); Industrial Cyber-Physical System (ICPS); Cyber Physical Production System (CPPS); Industry 4.0; Internet of Things (IoT); Wireless Sensor Networks (WSNs);**

I. Introduction

The work presented in this paper is a part of SelSus European Project, who's objective is the development of a diagnostic and prognosis environment, 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. The SelSus solution is composed of client "Sensor SelComps" running at sensor network gateway level and server "SelSus Cloud" applications. 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 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*) [1]–[3]. SelSus project [3], [4] is based on a conceptual framework called Sensor SelComp that combines an Industrial SelSus Cloud (ISC) with a Cyber Physical Production System (CPPS) aiming to enable the use of WSNs in the manufacturing domain, allowing WSNs digitization to be an integral part of CPPS. This framework is able to run in any operative system, fully modular and already compliant with the SelSus infrastructure. This means that any sensor integrated using this technology will be readily available for data visualization, access and processing in the SelSus Cloud. It is composed by a set of tools that highly increase its flexibility for any user to operate. These tools tackle mainly two aspects that trouble the manufacturing companies when dealing with sensors: 1) Sensor Integration; 2) Data Processing.

Sensor data is a key element for machine behaviour modelling and process optimization, where information can be gathered from machine parametrization (variables that control the process) and from its observable impact in the final quality of the product, allowing immediate monitoring of the process conditions. It can be used in a myriad of applications, from predictive maintenance, to the optimization of process to minimize the costs, never jeopardizing the product quality, and even knowing if a process is drifting from what was defined in the design phase. Currently WSNs, are composed of heterogeneous sensor nodes, implemented with different technology, a constraint that force developers to know low level details about protocols used by the sensors they are trying to integrate.

To tackle sensor data gathering automation and integration from heterogeneous WSNs, that send US-ASCII encoded data though serial port connected gateways. We had to develop a solution (described in Sections III and IV), that allows to easily create sensor data interpreters of raw data, to integrate, add or remove sensors through graphical interface in the SelSus Cloud, and readily deploy it to the corresponding Sensor SelComp, performing raw data decoding or interpretation. After this process, only the sensor needs to be physically integrated in the shop-floor, and the integration should be fully automated. Ultimately, such a technology enables the use and implementation of the plug'n'produce concept in WSNs for industry.

The rest of the paper is organized as follows. Section II, describes related work. In section III, we describe the flexible sensor integration followed by the description of our implementation in section IV, experimentation results in section V and then by results discussion in section VI. Finally, in section VII, the conclusion(s).

II. Related Work

In literature, Carter et al. [5], Dinh-Duc et al. [6], Ma et al. [7] translate a message structure into XML for encoding binary messages. The present work is an extension of this technique for US-ASCII encoded messages.

Cloud Manufacturing Service frameworks [8]–[12] are mostly based in the adoption of Internet of Things (IoT) technologies, where industrial clouds are heavily sensor based. The main role of Cyber Physical System for production systems, called Cyber Physical Production Systems (CPPSs) [13]–[17], is to realize intelligent monitoring and intelligent control, combining embedded systems, networks, sensors and actuators. Industrial Sensor Cloud (ISC) [18]–[22] supports CPPS by providing computational power, integration

in value and market chains, internal coordination and control, and external access through secure mechanisms.

Rajkumar et al. and Shi et al. [13], [15] explain what CPS are, their features, challenges and applications. Wu [14] explains how CPS applications exploit WSNs collected information to bridge real and cyber spaces and shows the WSNs network coverage and deployment issues. Stojmenovic [16] explores the Machine to Machine (M2M) concept in CPS and looks at futuristic applications.

Dillon et al. [18] give a Cloud computing overview while showing the challenges and issues of model adoption. Further challenges and issues towards model adoptions are described by Low et al. in [20]. Zhang et al. [19], describe the cloud architecture, characteristics with a review of technologies used in cloud computing environments. Sakr et al. [21], describe large scale data management approaches in cloud environments. Givehchi et al. [22], describe cloud computing technology for industrial automation, a general cloud model for automation is derived from existing proposals offering automation functions as services from a dynamic infrastructure.

In literature, we have not found combined ISC and CPPS with SelSus capability of sensor integration through graphical parser drawing. In the bibliographic revision described above, which presents the relevant industrial solutions, the whole integration is done via programming, so no such solution was found.

III. Flexible Sensor Integration

Our goal is to have a flexible Data Sensor Integration solution that allows to easily graphically develop an interpreter of the raw data packet in the SelSus Cloud and deploy it to execute embedded in Sensor SelComp at the WSN gateway level for automatic data acquisition. Java was the chosen programming language, because it's cross-platform supported, portable, has good library support, good performance and scalability due to HotSpot JIT compiler and native support for threads, type-safe and sand-boxed. Sensor SelComp were already implemented in Java, making the language the more logical choice. In binary, there is more than one way to encode a float, being one of them the standard IEEE 754. However, not all manufacturers use standard encoding. One of such example is Libelium [23] that in order to deal with comma placement determination uses non-standard encoding. So, we've chosen US-ASCII (or ISO 646) subset of UTF-8 encoded characters because it is a non-ambiguous encoding standard (only one way to encode and decode) for the message body.

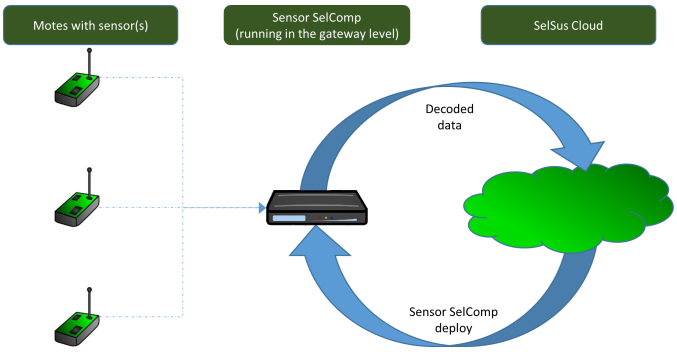


Figure 1: Interaction between: *Mote - Sensor SelComp (running in gateway level) - SelSus Cloud*

In Fig. 1, we show the interaction of *Mote - Sensor SelComp - SelSus Cloud*. Mote is a sensor node in a WSN that is capable of performing some processing, gathering sensory information and communicating with other connected nodes in the network. Sensor SelComp is the program that runs at gateway level of the WSN and functions as client of SelSus Cloud and is responsible for decoding WSN raw data. SelSus Cloud is where graphical interfaces are used to configure parsers and deploy it into the Sensor SelComp. SelSus Cloud is also responsible for data gathering, in-network processing and data visualization.

With the goal in mind, we developed two different approaches to solving the problem. The first approach is based in the use pattern match regular expressions both to detect and decode WSNs sensor data. The second use messages frame description to both detect and decode WSNs sensor data.

In the first approach, we have started by analysing the raw data of US-ASCII encoded messages using regular expressions as soon as we start receiving data from serial port, in order to classify or group characters and find patterns. In this analysis, we have made a few assumptions: 1) message size is at least 3 because we need to find at least SYNC, i.e., a sequence of chars that uniquely specifies the start of a message; 2) Punctuation chars are used to separate strings, excluding dot, comma and dash (separators) because this is also used in numbers. Based on this, we split the raw data (incoming US-ASCII encoded messages). Each split results in a list of strings. We start splitting by non-printable chars and in the process eliminate them. For each string in the remaining list, composed of 2 or more chars, we split it by separator chars. After this, all strings are ready to be classified into categories or groups: e.g., Numbers (Integer, Floats); separating chars (chars or group of chars used to separate blocks in a message, e.g., #); Words. After classification, we look for loop patterns (e.g., separating char - float - separating character - integer - separating character) as an indication of message structure, posteriorly validating or not the loop has a new message type discovery through a graphical interface. Each classified string is an item; the base unit we need for a correct message interpretation.

In Table I, we show a Libelium Waspmote IEEE802.15.4 indexed message frame example with battery measurement, where: the index counts the bytes or chars; STR is the message raw data string format; HEX is the message raw data hexadecimal encoded; and Fields or Items are the message units in which the message is divided.

Applying this approach to the example presented in table I. As a result we'd have two lists, one is the result of the string split and filtering [<=>, ?, #, 366369798, #, Battery, #, 0, #, BAT, :, 90, #, LC, :, 4.123, #] and the other is the pattern that represents this type of messages [String, Separator char., Separator char., Integer, Separator char., Word, Separator char., Integer, Separator char., Word, Separator char., Integer, Separator char., Word, Separator char.,Float, Separator char.]. Separating chars are needed because we can receive information of several different sensors in a Sensor SelComp. So, received messages can be different from each other and a fixed structure cannot be assumed. Also, items can be variable in size or length, e.g., integer and floats are not padded nor represented with the same precision. We have encountered some limitations in this approach like separating chars not being optional, making message encoding space consumption not ideal; If we want to expand support to binary encoding, or hybrid (binary and ASCII mixed), it would be very difficult to define classifiers.

Due to the limitations of the previously described, we've developed a new technology in a second approach that assumes the message frame content is known and the body is composed of US-ASCII encoded characters. Based on this, we defined that a certain message frame is composed by a set of items. The item is the atomic element that needs to be found or decoded in the messages. For each item, we define based on the indexed frame (see Table I example and

Table I: Indexed Message Frame Example from a Libelium Waspmote using IEEE802.15.4 and measuring battery voltage and percentage.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
STR	<	=	>	?	stx	#	3	6	6	3	6	9	7	9	8	#	B	a	t	t	e	r	y	#	0	#	B	A	T	:	9	0	#	L	C	:	4	.	1	2	3	#
HEX	3C	3D	3E	80	02	23	33	36	36	33	36	39	37	39	38	23	42	61	74	74	65	72	79	23	30	23	42	41	54	3A	39	30	23	4C	43	3A	34	2E	31	32	33	23
Items	sync			frame type	num. sensors	sep	serial id									sep	frame id							sep	frame seq.	sep	% tag				% val.		sep	volt. tag			volt. value					sep

description in page III): - a name; type; default value; start index; size; variable. The name is the identifier; the type (Integer, String, etc...) tell us how the data should be interpreted; default value is defined only if it has one; variable is a boolean that is false if an item has fixed size and true if the size of an item is variable (e.g. number are not padded). In order to start decoding a message, it was used a counter-based approach, where the items are analysed one by one by its arrival order, and therefore incrementing the counter until the message is totally analysed The start index counter starts at zero and counts until the end of the last fixed size item in order. If a variable size item is found in the parser when analysing a message, the counter is reset to zero. If we have variable sized items in the frame, we assume that after there is a fixed size item known. Normally, the first item is a SYNC which is a sequence of chars that uniquely specifies the start of a message. This kind of item is particularly important because it defines when to start interpreting the message. Additionally, other meta-data included in the parser is the manufacturer, model, serial port configuration (includes baud-rate, stop bits, data bits, parity, port name, message size (only if fixed sized)), physical link, protocol name and protocol norm, sensors and formulas (optional). Formulas are important to contemplate sensor raw data conversion into measurement units when the mote do not have the calculus precision needed or in cases where it is faster to convert it at gateway level. For each sensor we define: id; name; measurement type and units; manufacturer; model; serial number; minimum and maximum range; output type. For each formula we define, name, expression, variables name and units. In-order to extend decoding to binary or hybrid messages (with both binary and US-ASCII items), we simply add an extra field in item meta-data that indicates if the item is to be decoded from binary or from US-ASCII string. This presents all the meta-data needed to parse a message type coded in an XML file. We used XML, but equally valid solutions for storing this data are for example JSON and CSV.

IV. Implementation Of The Flexible Sensor Integration Solution

Fig. 2 shows the implementation sequence diagram of the proposed solution (second approach).

In the SelSus Cloud, a graphical interface (see fig. 3) is used to specify, create or alter a parser XML file. In the same interface, is possible to choose and deploy a parser to a Sensor SelComp that will start the interpretation of new sensor data. Once a created parser is deployed, the Flexible Sensor Integration solution running at the Sensor SelComp level: opens the parsers directory; read and unmarshall (creating an object representation of the parser) verifying parser file integrity through the use of a XSD file; indexes the list of parsers who share that same serial port configuration by configuration, then searches for serial ports available in the machine. If the serial port name is specified in the parser, it is only tested if the same port name is available in the machine. Otherwise, the parser is tested in all the available serial ports. In order to increase the performance of the system, the serial ports are tested in parallel, allowing multiple message analysis by the developed solution.

Foreach *configuration* and available serial port in the machine: tries to open the serial port with the configuration; for each parser in the list indexed by that configuration, determine the minimum message size needed in order to parse (sum of the size of fixed size items); and start reading raw data into a buffer. Each port has its own buffer. In parallel, for each parser in the list, the solution tries to match the first and last items, determining its indexes. If the indexes range size is greater or equal to minimum message size, the message decoding

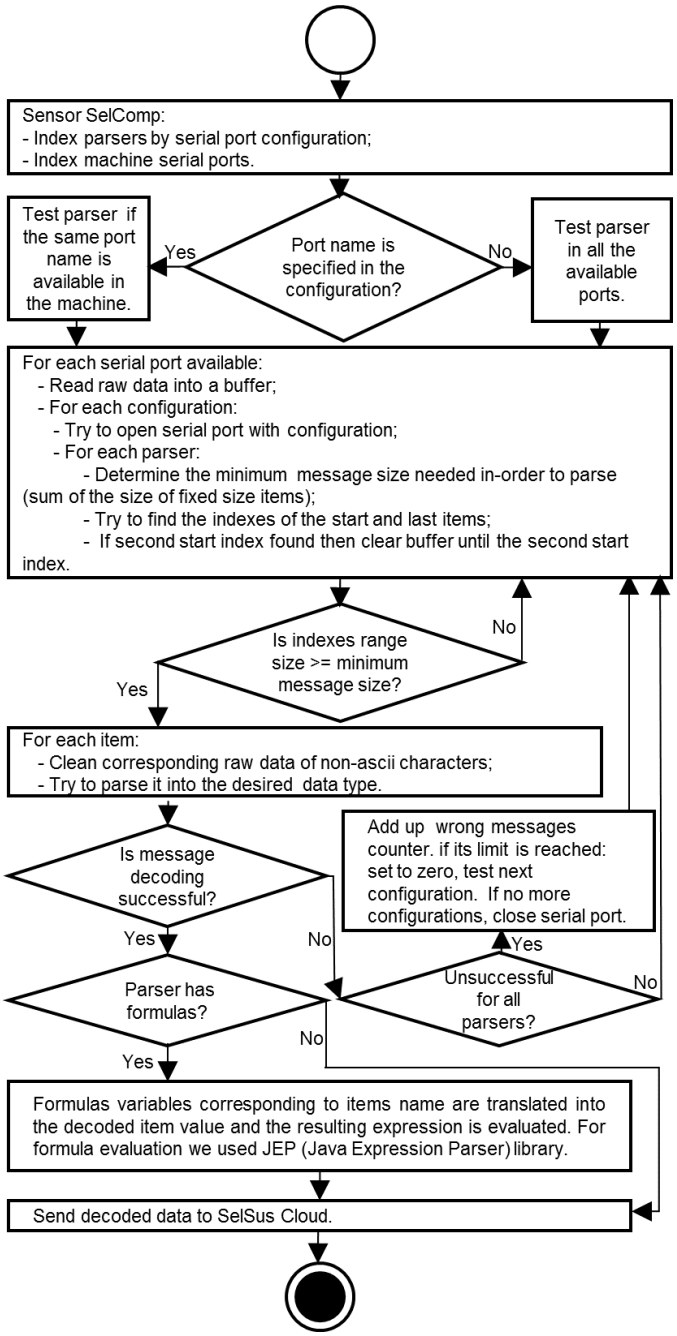
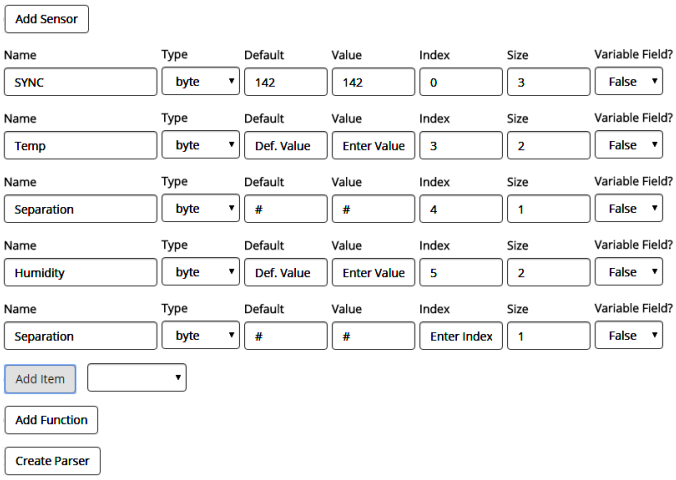
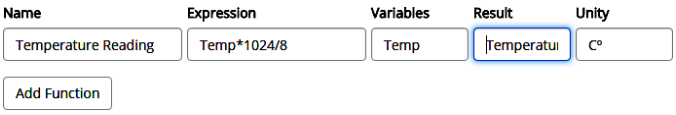


Figure 2: Implementation sequence diagram.



(a) UI to insert Meta-information to decode a ASCII message from Serial Port COM sensor integration.



(b) Function definition for the Parser.

Figure 3: SelSus Cloud User Interfaces (UI)

process is initiated.

Message decoding process tries to decode the message, matching the parser items one at a time: foreach item, the corresponding raw data is first cleaned of non-ascii characters and only then tries to parse into the desired data type. If the message decoding is successful and if the parser has formulas: formulas variables corresponding to items name are translated into the decoded item value in the expression and the result is evaluated. For formula evaluation we used JEP (Java Expression Parser) library. If a message decoding is successful: its decoded data is transmitted to the cloud, the buffer corresponding indexes cleaned and a counter for *Wrong Messages* is reset to zero. If message decoding process is unsuccessful for all the parsers, the *Wrong Messages* counter is added up. The buffer continues to be filled up and keeps trying to decode messages until the next start of a message is found. Once found the buffer is cleaned until that new start. If the *Wrong Messages* counter reaches its limit, it is set to zero and the next configuration is tested. If the end of configurations is reached, the serial port is closed.

V. Experimentation Results

The first approach (Pattern-match approach) was tested using as input for analysis a compiled 5MB data file of raw data acquisition from a wireless sensor network.

The second approach and present solution for sensor integration was validated using IEEE802.15.4 RF Protocol, with 4 motes (equipped with XBee Pro S1 XBP24-ASI-001J), 2 different baud rates (115200 Kbps, 57600 Kbps) and 4 sensors (Battery, ACC, PIR and Temperature). These tests were performed not only in a PC platform running Windows 10 O.S. (equipped with Intel Core i7-3630QM and 8GB of RAM), but also in a Raspberry Pi 2 running Debian 8.0 O.S. to test the efficiency in lower computational powered devices. Two major tests were made: a) 4 motes with one sensor each; b) 4 motes, but 1 mote with 1 sensor, 1 mote with 2 sensors, 1 mote with 3 sensors and finally 1 mote with 4 sensors each). The average message size in bytes in tests: a) sensor Battery is 42, ACC is 40, PIR is 28 and Temperature is 44; b) one sensor is 42, two sensors is 56, three sensors is 75 and four sensors is 84. The motes were powered through USB. The test environment was as close as possible to a real scenario, where interference is expected, in the presence of at least 28 IEEE802.11 APs and people moving. The exact same code and compiler optimization flags were used in both platforms.

The test results are presented in Fig. 4 and Fig. 5 in the format of histograms with yellow and green blocks corresponding to a numbers of messages parsed in a given amount of time. The blue and cyan lines are the log-normal distribution density curve fit to histogram. Both histograms were made using data sampled in a period of 1h with an acquisition rate of 0.2Hz per mote, corresponding to 2880 messages sent per hour. The time required to interpret a single sensor message is presented in milliseconds. From the Figures, we can see that the performance of the PC is better comparing with the Raspberry Pi 2, as expected.

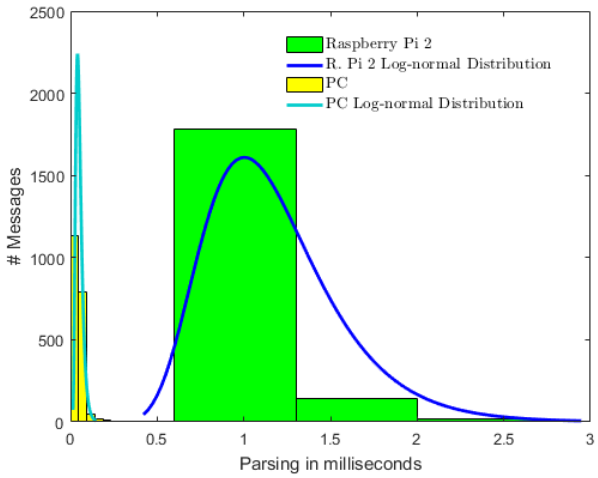


Figure 4: Acquisition Rate is 0.2 Hz with the duration of 1 hour. 1 sensor per mote. Baud rate is 57600 bps.

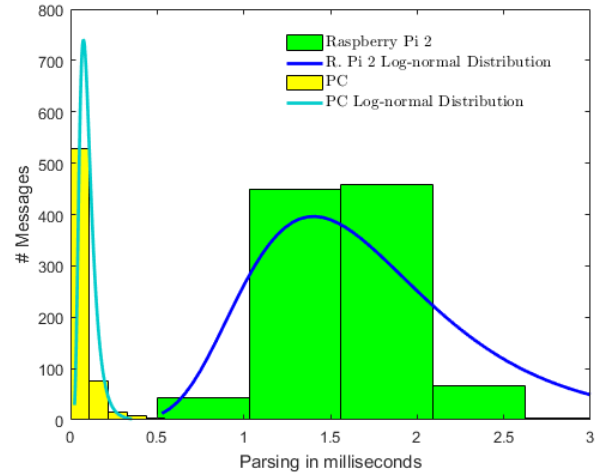


Figure 5: Acquisition Rate is 0.2 Hz with the duration of 1 hour. 1 to 4 sensors per mote. Baud rate is 115200 bps.

In Fig. 4, the results of test *a*) are presented, corresponding to Case 5 in Table II. Using the PC platform, the mean value of all decoded messages is 0.0551 milliseconds and standard deviation of 0.0909 milliseconds, on the other hand, using the Raspberry Pi 2, the mean value is 1.2373 milliseconds, with a standard deviation of 1.3071 milliseconds.

In Fig. 5, the results of test *b*) are depicted, corresponding to Case 3 in Table II. Using the PC platform, the mean value of all decoded messages is 0.1140 milliseconds and standard deviation of 0.1905 milliseconds and using the Raspberry Pi 2, the mean value is 1.7891 milliseconds, with a standard deviation of 1.4184 milliseconds.

Table II: Tests Results and Conditions.

Case #	Platform	Mean (ms)	STD (ms)	Frequency (Hz)	Baud-rate (bps)	Sensor/Mote
1	PC	0.1116	0.1806	0.2	57600	1 to 4
	R. Pi 2	2.2654	1.8794			
2	PC	0.1152	0.1678			
	R. Pi 2	2.4026	1.6727			
3	PC	0.1140	0.1905		115200	
	R. Pi 2	1.7891	1.4184			
4	PC	0.1146	0.1463			
	R. Pi 2	2.1323	1.7504			
5	PC	0.0551	0.0909		57600	1
	R. Pi 2	1.2373	1.3071			
6	PC	0.0580	0.0731		115200	
	R. Pi 2	1.2385	1.3107			
7	PC	0.1165	0.1023	1		
	R. Pi 2	2.1546	1.2834			
8	PC	0.0559	0.0644		57600	
	R. Pi 2	0.8435	0.9211			
9	PC	0.0470	0.2053		115200	
	R. Pi 2	1.0505	1.0591			

The Table II shows a summary of all the test results, presenting the mean decoding time and the standard deviation (STD) in milliseconds and also includes a description of the conditions or configuration represented by sampling rate, sensor per mote and baud-rate. All the test duration was 1 hour. The radio power settings for Case 2 and 4 to 7 were 10 dBm, in all others Cases were -6 dBm. Cases 8 and 9, correspond to the first tests made, where the non-ascii filter was not implemented. In order to try to improve the number of correctly interpreted messages further, the filter was implemented and used in subsequent test Cases (1-7).

VI. Discussion

During the tests, using a single gateway, we had a mean Packet Reception Ratio (PPR) of 61.99% with a standard deviation of 23.33%. In all cases motes were powered solely by USB. The tests were made with two different baud-rates to test synchronization difficulty, two (-6dBm and 10dBm) different radio power settings to test radio signal saturation. Additionally, it was tested the variation of message size (1 to 4 sensors) to test parser adaptability capability to cope with different messages at same time. In the worst case scenario, the platforms maximum message decoding times or recognition ranges from 0.1203 ms (case 8) to 0.3045 ms (case 3) in PC and from 1.7646 ms (case 8) to 4.1448 ms (case 1) in Raspberry Pi 2. This corresponds to the capability of decoding from 3284 to 8312 messages per second in PC and from 241 to 566 messages per second in Raspberry Pi 2.

For platform PC: tests a) the worst case scenarios ranges from 0.1203 ms to 0.2523 ms; tests b) the worst case scenarios ranges from 0.2609 ms to 0.3045 ms. For platform Raspberry Pi 2: tests a) the worst case scenarios ranges from 1.7646 ms to 3.4380 ms; tests b) the worst case scenarios ranges from 3.2075 ms to 4.1448 ms. In both platforms, we have an increase of decoding times, both minimum and maximum, as the message size increases.

Considering all the tests presented, in average the platforms maximum message decoding times or recognition ranges from 0.0470 ms (case 9) to 0.1165 ms (case 7) in PC and from 0.8435 ms (case 8) to 2.4026 ms (case 2) in Raspberry Pi 2. This corresponds to the capability of decoding from 8583 to 21276 messages per second in PC and from 416 to 1185 messages per second in Raspberry Pi 2.

This approach is advantageous in various scenarios. Therefore, two different scenarios will be presented to depict the inherent advantages of such an approach:

Scenario 1) The machine operator already has a wireless sensor network (WSN) integrated and adds a new sensor to an existing mote. In this scenario, using the SelSus Cloud graphical user interface, the operator only has to create a new parser or alter an existing one adding the new sensor to be interpreted using new items. Afterwards, the user can deploy the parser to the selected Sensor SelComp, and the new sensor integration is fully done.

Scenario 2) The machine operator has a machine working and he wants to monitor without affecting the production process. He first adds previously programmed motes to the machine with the required sensors, a single board computer running an instance of the Sensor SelComp acting as gateway for the WSN. Assuming the Chief of Operations knows the message structure sent by the motes, he can use the SelSus Cloud graphical interface, to create or add the parsers. Afterwards, the user deploys the parser to the selected Sensor SelComp and WSN integration is fully done.

VII. Conclusion

Our goal was to have a Flexible Sensor Integration solution that allows to graphically develop an interpreter of raw data packet, that executes at the gateway level for automatic data acquisition. Our hypothesis states that we were able to describe all the message meta-data in order to fully decode the message content, universally in a single file per message type, assuming the content is US-ASCII encoded and through a graphical interface. In order to test our goal, we programmed a wireless sensor network composed of 4 motes, created 8 different message types with up to 4 sensors information that we needed to be able decode. In our test-bed, we have achieved a message decoding capability of in-average of, 8583 to 21276 messages per second in PC and from 416 to 1185 messages per second in Raspberry Pi 2. That means an in average the decoding time spent per message is between 0.0470 ms to 0.1165 ms in PC and between 0.8435 ms to 2.4026 ms in Raspberry Pi 2. Based on the achieved results, we can state that the solution explored is applicable for monitoring the current state of the machine, using an acquisition rate down to 1Hz. These kind of acquisition rates are applicable when a monitoring of Temperature, Humidity, PIR, Pressure, Flow, CO, CO_2, O_2, O_3, etc. Validating our hypothesis and achieving our goal. In order to improve decoding time per message, in future work, we shall explore sensor integration directly in single board computers through SPI, GPIO, CAN and I2C interfaces using wireless standards for industry (e.g. WirelessHART and ISA100.11a) and this solution philosophy for real time data acquisition.

Acknowledgment

This research was supported by:

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] F. L. F. Almeida, "Designing and implementation of an intelligent manufacturing system," *Journal of Industrial Engineering and Management*, vol. 4, no. 4, pp. 718–745, 2011.

[2] G. Gonçalves, J. Reis, R. Pinto, M. Alves, and J. Correia, "A step forward on intelligent factories: A smart sensor-oriented approach," in *Emerging Technology and Factory Automation (ETFA)*. IEEE, 2014, pp. 1–8.

[3] L. Neto, J. Reis, D. Guimaraes, and G. Goncalves, "Sensor cloud: SmartComponent framework for reconfigurable diagnostics in intelligent manufacturing environments," in *13th International Conference on Industrial Informatics (INDIN)*. IEEE, jul 2015, pp. 1706–1711.

[4] "SelSus MANUFACTURING SYSTEMS." [Online]. Available: http://www.selsus.eu/

[5] B. Carter and R. K. Ragade, "Message transformation services for wireless sensor networks (mts-wsn)," in *ICWN*, 2006.

[6] A.-V. Dinh-Duc, "Using NViz tool for Environmental Sensor Networks," *ECTI TRANSACTIONS ON COMPUTER AND INFORMATION TECHNOLOGY*, vol. 7, no. 1, 2013.

[7] L. Ma, L. Wang, L. Shu, J. Zhao, S. Li, Z. Yuan, and N. Ding, "NetViewer: A Universal Visualization Tool for Wireless Sensor Networks," pp. 1–5, dec 2010.

[8] R. J. Ferreira, J. Machado, and H. Rose, "Cloud-based framework for advanced maintenance tasks," pp. 1324–1329, jul 2015.

[9] 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, pp. 1–14, feb 2015.

[10] X. Yue, H. Cai, H. Yan, C. Zou, and K. Zhou, "Cloud-assisted industrial cyber-physical systems: An insight," *Microprocessors and Microsystems*, vol. 39, no. 8, pp. 1262–1270, nov 2015.

[11] Fei Tao, Ying Cheng, Li Da Xu, Lin Zhang, and Bo Hu Li, "CCIoT-CMfg: Cloud Computing and Internet of Things-Based Cloud Manufacturing Service System," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1435–1442, may 2014.

[12] Qingping Chi, Hairong Yan, Chuan Zhang, Zhibo Pang, and Li Da Xu, "A Reconfigurable Smart Sensor Interface for Industrial WSN in IoT Environment," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1417–1425, may 2014.

[13] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-Physical Systems: The Next Computing Revolution Ragunathan," *Proceedings of the 47th Design Automation Conference*, p. 731, 2010.

[14] F.-J. Wu, Y.-F. Kao, and Y.-C. Tseng, "From wireless sensor networks towards cyber physical systems," *Pervasive and Mobile Computing*, vol. 7, no. 4, pp. 397–413, aug 2011.

[15] J. Shi, J. Wan, H. Yan, and H. Suo, "A Survey of Cyber Physical Systems," *Proceedings of the International Conference on Wireless Communications and Signal Processing*, pp. 1–6, 2011.

[16] I. Stojmenovic, "Machine-to-Machine Communications with In-network Data Aggregation, Processing and Actuation for Large Scale Cyber-Physical Systems," *IEEE Internet of Things Journal*, vol. PP, no. 99, pp. 1–1, apr 2014.

[17] I. Akkaya, P. Derler, S. Emoto, and E. A. Lee, "Systems engineering for industrial cyber-physical systems using aspects," *Proceedings of the IEEE*, vol. 104, pp. 997–1012, 2016.

[18] T. Dillon, C. W. C. Wu, and E. Chang, "Cloud Computing: Issues and Challenges," *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pp. 27–33, 2010.

[19] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: State-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, may 2010.

[20] C. Low, Y. Chen, and M. Wu, "Understanding the determinants of cloud computing adoption," *Industrial Management & Data Systems*, vol. 111, no. 7, pp. 1006–1023, aug 2011.

[21] S. Sakr, A. Liu, D. M. Batista, and M. Alomari, "A survey of large scale data management approaches in cloud environments," pp. 311–336, 2011.

[22] O. Givehchi, H. Trsek, and J. Jasperneite, "Cloud computing for industrial automation systems — A comprehensive overview," *18th Conference on Emerging Technologies & Factory Automation (ETFA)*, pp. 1–4, 2013.

[23] "Libelium." [Online]. Available: http://www.libelium.com/