Sensor Cloud: SmartComponent Framework for Reconfigurable Diagnostics in Intelligent Manufacturing Environments

Luis Neto, João Reis, Diana Guimarães, Gil Gonçalves
Institute for Systems and Robotics
Faculty of Engineering of University of Porto
Porto, Portugal
Email Address: {lcneto , jpcreis , dlguim , gil} @fe.up.pt

Abstract—**Sensor networks that consist of a variety of sensor nodes, with capability for distributed storage and analysis, interoperable and delay-tolerant communication, will pave the way for a truly scalable network of sensors which will support adaptable plug-and-produce assembly stations. The concept of Sensor Cloud has emerged as the cornerstone for enabling the integration of nearly real time data sources into Service Oriented Architectures and as one of the enablers for Reconfigurable Manufacturing Systems.**

Hitherto there are still some challenges that need to be tackled, namely the on-the-fly instantiation and update of services at the system level and the dynamic (re)organization of the services created. This paper presents the first steps in the development of a framework (taking advantage of several technologies like UPnP, OSGi and iPOJO) to address these challenges and make Sensor Clouds a reality in the shop floor. The results from the first implementation reveal that the performance of the system is linear in terms of scalability, and from these scalability tests, we proved the framework's robustness and consistent responsiveness.

I. INTRODUCTION

Taking into account the possible level differentiations that can exist in a Manufacturing System, the present work focus its implementation on the Logical level, where the most complex and agile functionalities should reside. Those functionalities tackle problems like easy integration of equipment as *UPnPDevices* – UPnP Architecture; easy and dynamic model instantiation for data processing (based on the software components available, the system allows as many model instantiations as desired) and on-the-fly replacement of those software components for updated versions - *OSGi Apache Felix Framework*; and dynamic reconfiguration of the logical architecture (rearrange the dependencies between the model instances and also the inputs of the system) - iPOJO. This way the platform can guarantee the flexibility required for data monitoring and handling, and correct use of information in levels very close to the shop-floor equipment.

The basis of this work, the *SmartComponent Framework* is responsible to collect, accommodate, treat, process and disseminate sensor data. Those functionalities have a direct impact in the requirements of the latest concepts of Sensor Cloud, and consequently, in the concretization of important industrial concepts like Reconfigurable Manufacturing Systems (RMS). The main purpose of Sensor Cloud is to easily integrate sensors into the system, manipulate, access and visualize sensor information using Cloud-based technologies [1]. One of the latest explored applicability of Sensor Cloud in the industry is related with the monitoring and diagnosis of machines as key factors for achieving the main characteristics of RMSs. Thereby, the use of Sensor Cloud can be viewed as an enabler for RMS, or more concretely, as a Reconfigurable Inspection Machine (RIM) for machine easy monitoring, process and visualization, which is clearly explored in the present paper.

The present work has been developed in the context of an European project called Self-

Sustaining Manufacturing Systems (SelSus). Its focus is on machines health monitoring, by analysing a large amount of data gathered from the shop-floor and helps to predict possible malfunctions and infer its root cause.

This paper is organized in 4 main sections. *Industrial Context*, where an industrial context about Sensor Cloud and RMS is given. *Framework*, exposing all the Framework's implementation details and functionalities. *Results*, presenting all the tests performed to validate the framework's execution. Finally, *Conclusions and Discussion*, where the potential impact of the framework is explored and presents some future prospects to be exploited.

II. INDUSTRIAL CONTEXT

A. SensorCloud

The Sensor Cloud concept emerged as an extension of Cloud Computing concept (see [1] for details) to allow device management. It provides a vast storage capacity and processing capabilities and has various advantages over the direct sharing. The Sensor Cloud virtualizes multiple physical sensors making it possible for users to benefit from the sensors and control them - monitor availability and status, set a frequency of data collection and group different sensors - via their Web browsers without worrying about their locations and detailed specifications. The literature shows mainly a three layers Cloud structure - *Sensor Integration Layer*, *Logical Layer* and *User and Application Layer*. The publish-subscribe mechanism gives the End-user the customization requested. Some related works are: [2] [3] [4].

In the context of this work, the sensor integration would no longer be a concern, once the *SmartComponent* would take this responsibility. The Cloud will interact with the sensor's virtualization at the *Logical Layer*, where the services' management takes place (with a service logic reconfigurability level similar to the one *SmartComponent Framework* have), making the subscribed information available to the consumers. Based on this logic, the presented concept and dynamics of Sensor Cloud can be viewed as an enabler for the inspection process of a Manufacturing System, where the shop-floor machinery should be monitored in order to infer in run-time its current condition and assess its performance.

B. Reconfigurable Manufacturing Systems

Since the 1980s, the production paradigm has diverted its attention from mass production to mass customization, due to the possibility have the same production costs in both approaches and the latter being closer to the customer demands, presenting a wider range and spectrum of product variety [5]. All these facts led to the concept of RMS where, in comparison with Dedicated Manufacturing Lines (DML) and Flexible Manufacturing Systems (FMS), the greatest advantage is the system's responsiveness attending to sudden market changes combining the high productivity of DMLs together with the flexibility of FMS [6].

Most of the RMS characteristics - *Customization*, *Convertibility*, *Scalability*, *Modularity*, *Integrability* and *Diagnosability* - are basically hardware goals to be met on both production and system level. As presented in the previous subsection, the inclusion of Sensor Cloud concept on the manufacturing process is one way of achieving the inspection and monitoring of machines in an easy, flexible and reconfigurable way – similarly as a Reconfigurable Inspection Machine (RIM) - fulfilling the *Diagnosability* characteristic, together with *Integrability* and *Scalability* (in terms of sensors). In case of sudden changes in the market, the use of Sensor Cloud allows for a quick and robust change in both hardware and software RIMs (as a Sensor Network), shortening the ramp-up phase when comparing with DMLs and FMSs, by means of a easily deployable and reliable solution.

Additionally, I-RAMP[3] is an ongoing European Project that takes advantage of virtual representation of shop-floor equipment (machines and sensor) for easy logical reconfiguration using task-driven communication and sensor data validation techniques. For more details see [7].

III. FRAMEWORK

This chapter presents all the technologies used to build the proposed solution. For each one, a brief description is given, along with all the details of the internal logic of the framework and its internal components.

A. Technologies

1) OSGi Apache Felix Framework: OSGi technology provides a controlled execution environment and orchestrates the interactions between specially built modules of code, known as bundles. A bundle is a plain old JAR file, which is manipulated during the build process, and includes a *MANIFEST.XML* file declaring its public, protected and private dependencies. [8]

Bundles run in the adopted OSGi framework execution environment (Figure 1). A required bundle to be deployed in the framework must embed or declare its own dependencies, because bundles have no a-priori knowledge of each other and dependencies must be resolved by the framework if not embedded.

The Apache Felix framework was chosen because of its extensive documentation, and also for including an UPnP specification implementation plug-in, officially integrated and developed by the Apache Felix project [9]. To understand the framework functionalities and consolidate the special considerations in developing OSGi application.

2) UPnP Basedriver: Acting as a bridge between the framework and the network, this driver [10] allow to import and export UPnP devices. Represented in Figure 1 in red, it is composed of two internal modules that provides two core services - the *Exporter* and *Importer* module. The *Exporter* module registers objects implementing the *UPnPDevice* interface specified by the OSGi specification. Any changes in that device (*OSGi ServiceEvents*) will be reported to the framework. The *Importer* module, that is listening in the framework for the changes, will consequently export and reflect these events to the network. Events coming from the network to the exported device will follow the opposite logic path. They are notified to the framework and an *UPnPDevice* instance listening to these service events will be matched through its unique *ServiceProperties* and notified of external actions. This core module implements the *ControlPoint UPnP* specification, which allows start listening *ServiceEvents* from *UPnPDevices* to the framework.

3) iPOJO: iPOJO is a service oriented component model for modular application systems, strongly integrated with the Apache Felix OSGi framework. The focus of iPOJO framework is to orchestrate the interactions between the components of a SOC (service oriented computing) application based on the separation of the code implementing the service functionality from the SOC coupling dynamics.

The iPOJO component is represented in Figure 1, in the same colour used to represent the interactions between the components that it is responsible to manage (Figure 2). The use of this component allowed us to focus just in the business logic and functionality of the application components. Hence, the components of the system still remain *"plain old Java objects"* (POJO). [11] The result is an adaptable application, regarding loosely coupling of services, in which, the internal constitution can be rearranged, depending on the deployed components in the execution environment and the shop-floor device composition.

B. Logical Architecture

The components in Figure 1 have the main responsibility of driving the architecture to its final purpose: data analysis, management, reconfiguration and logistic of services. The developed API defines two internal services to retain: *DeviceServices*, which represent a service provided by a device (eg: Temperature from a sensor or Position of a robotic arm); *ComplexServices*, which represent the services provided by instances of analysis modules deployed in the *SmartComponent* (eg: *Aggregation Mean* and *MinMax Validation*).

As OSGi bundles, every component provides at least one service at OSGi framework level. Provided services are ready to be consumed as soon as all its dependencies (Figure 2) are resolved by the OSGi framework. If a service leaves, all consumers depending on that service must be notified and proper actions must be taken. iPOJO plays a preponderant role in managing this volatile behavior. It provides callback methods for (un)registering, stopping or updating a service, thus ensuring the application has a much more controlled and stable behavior.

Internal functionality of the framework relies on a SOC approach [12]. The services at the *SmartComponent* framework level are extensions of the *SmartComponent Service* abstraction, defined by
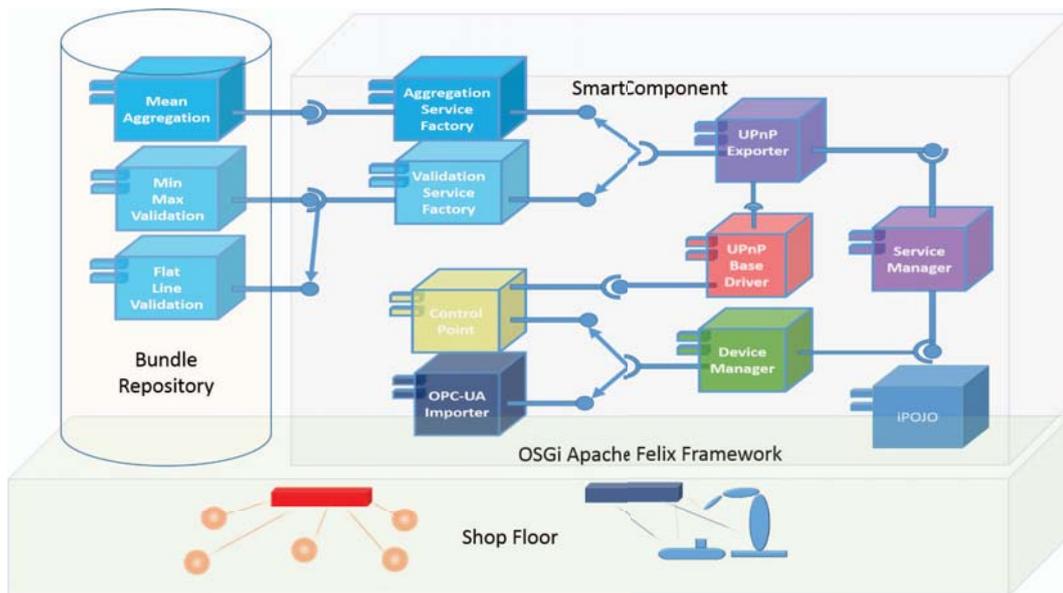
Figure 1.   Architecture Components Diagram.

the API. Recurring to the *uniqueness of prime numbers* [13] a strategy for identifying each service to grant uniqueness and efficient hierarchical tree organization has been implemented.

DeviceServices are introduced in the framework by modules providing a service compliant with the *DeviceManager* module. This module listens in the framework for bundles providing *DeviceServices* specification. As soon as the *ContolPoint* and *OPC-UA Importer* bundles are active, the *DeviceManager* starts listening for events from that components. An event can be a device (un)registration or a message containing data from a service of a specific device. This way, new modules of communication (eg. *ModBus, Profibus or iLan*), could easily be integrated to make the system adapt to incorporate new types of hardware.

Figure 2.   Component interaction description.

C. Framework Functionalities

To interact with the framework, components are exposed to the network through the *UPnP BaseDriver*, as *UPnPDevices*. Three components are immediately exposed as soon as the *SmartComponent* is running. The *ServiceManager* component allows retrieving information and dispose the active services in the framework, and the two *ServiceFactory* components allows to instantiate new *ComplexServices* and retrieve information about the analysis models that each one can instantiate.

1) Analysis Modules Deployment: ServiceFactory components (Figure 1 in light blue), are like containers for analysis modules producing instances of *ComplexServices* by type; as said before, a *ComplexService type* can be *Aggregation* or *Validation*. To manage the *type* association to the correspondent *ServiceFactory* and perform instantiation, two software pattern strategies are combined, *Whiteboard* [14] and *Factory* patterns.

2) ComplexService instantiation: As *UPnPDevices*, the factories modules provide the following functionalities: ***InstantiateServiceAccumulationCycles*** - Allows the instantiation of a new *ComplexService*, where the service providers can

1709

also be specified; ***InstantiateServiceFrequency-Data*** - It has the same functionality of the previous action, but additionally it allows to specify the time in milliseconds in which the information should be collected from the service providers; ***ListAvailable-Services*** - Indicates the services that the factory is able to instantiate. Once a service is instantiated, it is registered in the *ServiceManager* module and a correspondent representation is exported to the network by the *UPnP Exporter* module.

3) Services Reconfiguration: As the services met at the same level of abstraction, they can be (de)coupled easily. This versatility allows an on-the-fly rearrangement. The UPnP representation of each *ComplexService* provides 4 *UPnPAction* methods to: add consumers of the service's output (***AddConsumer***); add providers of information as service's input that can be either *DeviceService* or *ComplexService* (***AddProvider***); remove providers of information as service's input (***RemoveProvider***); and finally to list all the existing providers of information (***ListProviders***).
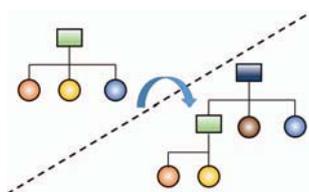

Figure 3.   Reconfiguration of services on-the-fly.

In Figure 3, circles are *DeviceServices* and rectangles *ComplexServices*. In the first scenario a complex service is consuming data from 3 device services. Analyzing the differences, a clear reconfiguration is illustrated in the second scenario. A new complex service instance is consuming from the previous complex service, the blue sensor service was disconnected from the initial complex service and added as provider of the newer one. Finally, a new device service, the brown circle, was introduced and selected as provider of the new instance.

IV.   RESULTS

Based on a simulation of a demonstrator case of the European project SelSus, the framework has been tested formulating a hypothesis. One of the sub-tasks of an assembly unit is the application of sealant in the contact surface of two parts. The sealant is applied by a robotic arm equipped with 4 sensors and the parts transported in a NC-Axis, which incorporates 2 sensors. The set of machines embeds a total of 6 sensors, from which data needs to be collected and treated.

To simulate the sensors, a tool was built to announce the sensors in the network as *UPnP Devices* through a dedicated wired *Ethernet* network. The generated data was threated by three developed *ComplexServices*: *MinMax* fires warnings for measures that exceeds minimum and maximum thresholds for each specific type of sensor service (eg. Temperature, Pressure); *MeanAggregation* is a mean calculation of the given sensors input; and *DummySum* produces a sum of all the inputs. The interaction with the framework and the response times were collected by the *UPnP Device Spy* tool [15]. The tests were performed by calling the following method, exposed by the *UPnPDevice* representation of each *ComplexService* instance running in the framework: ***GetSnapshot()*** - generates a new result.

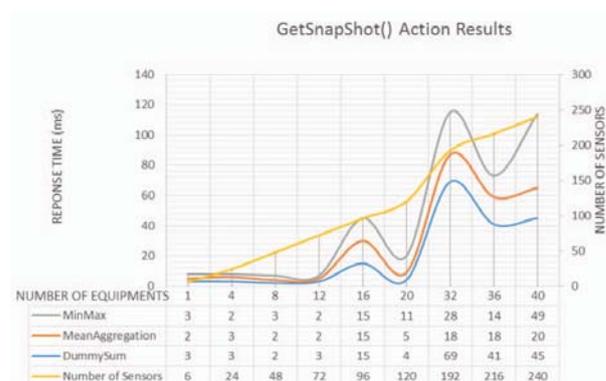| NUMBER OF EQUIPMENTS	1	4	8	12	16	20	32	36	40
MinMax	3	2	3	2	15	11	28	14	49
MeanAggregation	2	3	2	2	15	5	18	18	20
DummySum	3	3	2	3	15	4	69	41	45
Number of Sensors	6	24	48	72	96	120	192	216	240

Figure 4.   Response time performance

A first test displayed a linear response time until an exponential increase was recorded with **144** sensors. A sudden exception closed the *HTTP Socket*, and therefore, a second test was performed. The second test involved three powerful machines, contrary to the two in the first test; one containing the *SmartComponent* and the other two running the simulation environment. The machines running the simulation created a total of **240** sensors. The

results observed in the second test are displayed in Figure 4. The response times showed a linear trend and the exponential behavior that was previously observed, was due to the non-capacity of the hardware to handle a large number of parallel connections.

V. CONCLUSIONS AND DISCUSSION

The technologies and concepts explored and depicted in this paper reflect some of the Framework's logical architecture and functionalities that can be used for the achievement of Sensor Cloud approach in terms of its Logical Layer, providing already the tools and techniques for sensor diagnostics. Moreover, in the industrial context (specifically in the RMS), the concept of Sensor Cloud concept reveals to be an efficient and effective software solution for all the logic involved in monitoring, accessing and dynamic organization of equipment virtual instantiation and models as integrated software components. Dynamic inclusion, removal and update of software components for data treatment based on the equipment virtualization; automatic logical architectural reconfiguration based on the available model dependencies; and model replacement on-the-fly in the system are the foundations of the work in the present paper.

Regarding the results of the present architecture, the tests have proven a correct function of the system. Response to action invocation has a linear temporal growth and, not being yet totally independent from the number of sensors being monitored, the increase of processing time is so small (lasting milliseconds) that it should be considered independent in pragmatic perspective. As the cost of hardware is in constant decrease, we conclude the solution is feasible and covers all requirements of the simulated application scenario.

REFERENCES

[1] M. Yuriyama and T. Kushida, "Sensor-cloud infrastructure - physical sensor management with virtualized sensors on cloud computing," in *Network-Based Information Systems (NBiS), 2010 13th International Conference on*, Sept 2010, pp. 1–8.

[2] A. Alamri, W. S. Ansari, M. M. Hassan, M. S. Hossain, A. Alelaiwi, and M. A. Hossain, "A survey on sensor-cloud: Architecture, applications, and approaches," *International Journal of Distributed Sensor Networks*, vol. 2013, 2013.

[3] M. Eggert, R. Häußling, M. Henze, L. Hermer-schmidt, R. Hummen, D. Kerpen, A. Pérez, B. Rumpe, D. Thißen, and K. Wehrle, "Sensorcloud: Towards the interdisciplinary development of a trustworthy platform for globally interconnected sensors and actuators," in *Trusted Cloud Computing*, H. Krcmar, R. Reussner, and B. Rumpe, Eds. Springer International Publishing, 2014, pp. 203–218.

[4] P. Zhang, Z. Yan, and H. Sun, "A novel architecture based on cloud computing for wireless sensor network," in *Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering*. Atlantis Press, 2013.

[5] S. Hu, J. Ko, L. Weyand, H. ElMaraghy, T. Lien, Y. Ko-ren, H. Bley, G. Chryssolouris, N. Nasr, and M. Shpitalni, "Assembly system design and operations for product variety," *{CIRP} Annals - Manufacturing Technology*, vol. 60, no. 2, pp. 715 – 733, 2011.

[6] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy, and H. Van Brussel, "Reconfigurable manufacturing systems," *CIRP Annals-Manufacturing Technology*, vol. 48, no. 2, pp. 527–540, 1999.

[7] G. Goncalves, 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), 2014 IEEE*, Sept 2014, pp. 1–8.

[8] OSGi, "Osgi aliance," ""http://www.osgi.org/Main/HomePage", 2015 (accessed February 13, 2015).

[9] T. A. S. Foundation, "Apache felix framework," ""https://felix.apache.org/", 2015 (accessed February 13, 2015).

[10] ——, "Apache felix upnp basedriver," ""https://felix.apache.org/site/apache-felix-upnp.html", 2015 (accessed February 13, 2015).

[11] C. Escoffier, R. S. Hall, and P. Lalanda, "ipojo: An extensible service-oriented component framework," in *Services Computing, 2007. SCC 2007. IEEE International Conference on*. IEEE, 2007, pp. 474–481.

[12] M. Huhns and M. Singh, "Service-oriented computing: key concepts and principles," *Internet Computing, IEEE*, vol. 9, no. 1, pp. 75–81, Jan 2005.

[13] K. Thoelen, N. Matthys, W. Horré, C. Huygens, W. Joosen, D. Hughes, L. Fang, and S.-U. Guan, "Supporting reconfiguration and re-use through self-describing component interfaces," in *Proceedings of the 5th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks*. ACM, 2010, pp. 29–34.

[14] P. Kriens, "Osgi design technique," ""http://blog.osgi.org/2006/04/details-are-important-when-you-are.html", 2015 (accessed February 13, 2015).

[15] Intel, "Developer tools for upnp technologies," ""https://software.intel.com/en-us/articles/intel-tools-for-upnp-technologies", 2015 (accessed February 13, 2015).