A Component Framework as an Enabler for Industrial Cyber Physical Systems

Luis Neto*‡, Anders L. Madsen†§, Nicolaj Søndberg-Jeppesen†, Ricardo Silva*‡,
João Reis*‡, Peter McIntyre¶ and Gil Gonçalves*‡
*ISR-P, Instituto de Sistemas e Robótica, Porto, Portugal
†HUGIN EXPERT A/S, Denmark
‡FEUP, Faculdade de Engenharia, Universidade do Porto, Porto, Portugal
§Department of Computer Science, Aalborg University, Denmark
¶Ford Motor Company, Powertrain Manufacturing Engineering Europe

***Abstract*—In this paper we show how Component Based Software Engineering (CBSE) concepts were applied to design the Sensor SelComp. Therefore, a component framework to address the abstraction of Component Level within the SelSus European Project ICPS. We show how the framework can be used to tackle IPCS virtualization of sensors, machines and data processing. Moreover, we show the applicability of Sensor SelComp to a real case scenario where a Bayesian Network model was applied to a SelSus demonstrator to perform runtime fault detection in a RTV dispenser machine from Ford Motor Company.**

***Index Terms*—Industrial Cyber Physical Systems; Component-Based Software Engineering; Bayesian Networks; Smart Component; Smart Manufacturing**

I. Introduction

Modern production lines benefit from the IoT (Internet of Things) and ICPS (Industrial Cyber Physical Systems) technological advances to create environments where Smart Connected Products can influence its own production process and companies can benefit from Service Based business models. To accomplish this, there is an essential driver, information; the differentiating aspect regarding it nowadays are the sources and the amount of it. The number of internet connected devices combined with the granularity of information produced by a single device is growing so much that the estimated volume of data just in industry reached 1000 Exabytes annually and it is expected to increase by a factor of 20 times. [1]

This work is devoted to improve cooperation of field devices and industrial plant personnel therefore fostering the smartness in a Smart Factory environment. According with [2], any actor in a production plant can be considered a manufacturing component (e.g. computers, sensors, production machines, PLC's, robot's and humans). Once an actor in the production line is digitally abstracted by the factory ICPS, it becomes a Smart Manufacturing Component (SMC). This abstraction provides a physical component with smart capabilities, which enables it to become an active part in the Smart Factory ICPS, and consequently in the production environment.

The work presented in this paper is part of SelSus European Project, whose objective is the development of a diagnostic and prognosis environment. The SelSus ICPS defines three levels of abstraction for its constituents: 1) Component Level, which relates directly to machines or its sub-components and is composed of smart sensory capabilities, methods for self-diagnostics and predictive maintenance; 2) Station Level, at this level the developments are constituted by previous capabilities plus human machine interface and tools to support the design and maintenance of the factory station; 3) Factory Level, previous levels capabilities are combined to create a semantic driven maintenance scheduling for large production factory plants.

In a ICPS arrangement (e.g. in the Manufacturing Pyramid), Component Level is mostly related to hardware and mechanical properties of a production system. To make digital twins of these components, one of the difficulties generally found in any shop floor is the incompatibility between communication stacks, because industrial grade solutions most of the times are vendor specific. Another problem, is that machines and its sub-components also lack the necessary sensory information to build and feed feasible fault detection models. Lastly, under an ICPS federation, these digital abstractions must be semantically cohesive so any analysis or control module can interface them. These issues require a reconfigurable and updatable SMC too keep up with the always changing production system composition and configuration. An SMC must provide consistent mechanisms with which to configure, deploy and dynamically reconfigure code; which is a problem addressed by Component Based Software Engineering (CBSE) discipline.

In the rest of the paper we show how CBSE theory (Section II) was combined with the SMC concept to create the *Sensor SelComp* component model and framework (Section III). In addition (Section IV), we show an application of the Sensor SelComp to add external sensors to an RTV Machine in order to feed and run a BN model created to perform live fault detection and inform it to the ICPS. We conclude (Section V) by making considerations to improve the BN model used and the Sensor SelComp.

II. Component Models

To achieve software reconfigurability, in this case, the capacity to plug-in software components, and this way rearrange a system, a component model and framework are required. A software architecture designed by a component model solution

is developed as *"a composite of sub-parts rather than a monolithic entity"* [3].

Briefly, in a software system composed by components, a component model is a set of rules, conventions and standards. It dictates how a component is defined, interfaced, tested and deployed for execution. In practical terms, the component framework is the set of tools that implements the component model specifications. The framework is an aggregation of a mandatory runtime and optional tools, such as repository, compiler, verification and composition environment. The component model is the foundation of a component based design. It defines the composition standard, that is: how components are composed into larger pieces, how and if they can be composed at design and/or runtime phases of component life-cycle, how they interact, how the component repository (if any) is managed and the runtime environment that contains the assembled application. Because all of this, component models are hard to build, some problems like: achieving determinism and real-time, parallel flows of component and system development, maintaining components for reuse, different levels of granularity[4] and portability [5].

In terms of application to industrial environments, a component model eases the task of building control and monitoring applications. Considering a visual composition environment, an user which is not familiarized with software development, can simply design applications by pick and place, e.g., data processing algorithm, machines and sensors representations. The person in question does not need to know the internal details of a component, all it sees is a black box whose functionality, configuration and interface is formally and well documented. There is a clear separation between component technology applied to business and embedded systems, which is the case of interest to the lower levels on an ICPS. Embedded systems concerns range from scare physical resources, Quality of Service (QoS), timing to trustworthiness properties (security, safety and reliability). The latter are of special importance regarding safety critical applications. The following properties are the foundation of any Component Model.

a) Component Implementation and Interface: Components syntax, is the language used for component definition and which may be different from the implementation language. Depending on the component model, implementations can be binary, byte code, compilable code (C, C++) or custom languages. Properties and functionalities that are externally visible to a component constitute its interface.

b) Component Contracts: Interfaces show what a component does without telling how, and so, they are the very first form of component verification and validation. The relation between client and provider components is not the unique dynamic that must be agreed and often components must express constraints and quantitative properties. Components that need to declare QoS properties, depend on extra system layers such as the OS or runtime environment. Thus, contracts need to capture functional and extra-functional properties in component interfaces. Beugnard et al. [6] propose four levels of contracts, ranging in its flexibility to be negotiated by the component runtime.

c) Component Composition: The composition environment supports the developer in several steps of the development process. It is out of scope to think in how components are and how to define interfaces and properties. Therefore, the composition environment comes into play when the developer constructs, tests or deploys an application. In application construction the developer can either implement the inner functionality of a component and wire components to form an application. Depending on the component reuse policy, the environment can allow to use already existing components to build a new component. Testing comprises test case generation, debugging and test case execution. The deployment phase must support the installation of an application in a target device. Components can be composed using wrapping, static and dynamic linking, and "plug-and-play"; according with the composition environment.

d) Component Base System Life-cycle: In software engineering there are several models to guide the development life-cycle [7]. The *waterfall* is a classic sequential model in which each phase must be complete before proceeding and the output of a phase is the input of the next. These models can be adapted to conceive component based life-cycle.

III. Sensor SelComp

The SelComp (SelSus Component) is a realization of the SMC concept, which in practical terms is a digital shell that cohesively abstracts entities under the same ICPS federation. In the SelSus ICPS domain, there are two types of SMCs, the Sensor SelComp and the Machine SelComp. The first [8] is used to: a) abstract machines that lack any useful technology that allows to virtualize them (e.g. purely mechanical); b) to abstract sensors coupled to machines whose built in sensory information is not enough to feed the data analysis methods requirements; c) to abstract sensor networks deployed in the shop floor to capture physical properties of the environment. The second is used to abstract assets that have all the technological capabilities required to be abstracted as a SelComp.

It can be said that this SMC concept was first introduced by Multi-Agent Systems applied to production devices [9], and later adopted by the ICPS [10] to realize the "digital twin". The idea behind the Sensor SelComp was to have a piece of software that could act as generic abstracting shell. To accomplish this, the main obstacle was the communication. To solve that, the Sensor SelComp should be capable of extending its capabilities in runtime to support the required protocols to abstract a certain entity. The obvious answer to this problem was to model the Sensor SelComp as a Component Model Middleware, with that, even local data processing could be added and reconfigured in runtime, all this transparent to be controlled by the ICPS on demand.

The realization of the proposed component model and framework was guided by theory discussed in Section II. To better comprehend the implementation details, one may refer to Table I. The properties shown were selected based on the

Component Model		Component Framework	
Language Syntax and Semantic	Java Class, Java Object, Java Interface	**Code Development**	Any Java IDE, Text Editor
Interface	Native and Custom Data Types	**Component Compiler**	Maven 3 [11] (JAR Package, Java Compiler), Rhino [12] (Compile Java Script to Java)
Implementation	Java and JavaScript Code, Byte Code	**Component Deployment Format**	JAR File Package
Contract, Level 1	Java Interface, Shared Memory Space, Sockets	**Repository**	Maven Private and Public Repositories (For Design Phase Composition)
Contract, Level 2	XML Component Descriptor with I/O and Parameters Ranges		Sensor Cloud Framework (For Deployment and Runtime Composition, Figure 1)
Contract, Level 3	Sequential Publish/Subscribe	**Runtime Environment**	Java Virtual Machine
Contract, Level 4	Not Implemented (Best Effort)	**Composition Environment**	Java IDE for Design Phase with Explicit Dependency
Composition	Design Phase, Deployment Phase and Runtime		Sensor Cloud Framework for Deployment and Runtime, Figure 1

TABLE I
SELCOMP COMPONENT MODEL AND FRAMEWORK DETAILS

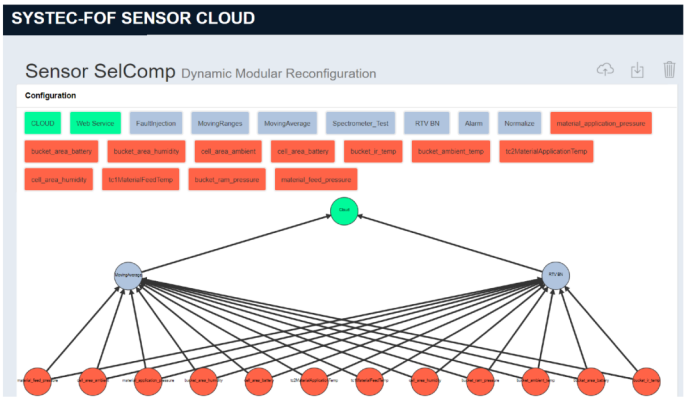

Fig. 1. Sensor Cloud Composition Environment.

taxonomies proposed in the works of Lau and Wang [13], [3] and Crnkovic et al. [14] to elucidate the proposed solution and for simplicity of comparison with others. As to concern with similar solutions, works [15], [16] present a series of component models for embedded systems in industry based in software engineering best practices. Concerning with differentiation aspects, this solution was developed to be generically integrated in any ICPS ecosystem, which by itself constitutes an improvement to the current state of the art. Delving into details, this component framework has the advantage to enable the plug in of different communication protocols and information processing algorithms during design, deployment and runtime phases. Moreover, any sensor or machine abstracted by this component model, is in line with the concept of *Administrative Shell* proposed in the *Reference Architecture Model for Industry 4.0 (RAMI 4.0)* [17].

A. SelComp Components

The SelComp components were split in three main categories, according with communication and data processing needs of an ICPS. The Sensor Cloud interface (Figure 1) depicts the different classes of modules in distinct colours. Rectangles represent modules that can be instantiated and circles are instances of a specific rectangle. The information flow is organized as a Directed Acyclic Graph and can be rearranged in runtime using a pick and place design interaction. The blue rectangles and dots in Figure 1 were used to abstract and run a BN model and a Moving Average, both used in the case of Section IV. In Figure 1 the ICPS interface components (orange circles representing the sensors in Fig. 3) were used to send the Data Processing components (blue dots), which by its turn send their output to the SelSus cloud system (green circle).

IV. APPLICATION IN SELSUS

To meet the challenges of reducing costs, speeding up product launches and creating line flexibility, Ford have moved away from traditional gaskets for many oil sealing joints and opted for a liquid sealant solution known as RTV (Room Temperature Vulcanization). This is true across the automotive industry. RTV applications are used on the engine assembly lines in single path flow, meaning they are on the critical path for production in all Fords powertrain plants, and part of a facility which quickly generates overhead costs when it is not producing engines. Around 200 operators will cease working if an RTV machines fails. The business case for moving away from gaskets gave production plants a challenge in maintaining a sensitive, complex machine within a production environment.

These machines have proved to be constraints for assembly lines largely because of poor First Time Through (FTT). The machines rarely fail, but don't have sufficient monitoring within the sub-systems to detect a change in the characteristics of the material, and the quality of the output. They can continue producing deteriorating sealant beads unless manually stopped, and the effort in repairing a poor application is time-consuming and a quality risk.

The goal is to stop RTV (Room Temperature Vulcanization) machines applying faulty beads altogether by detecting issues earlier within the material delivery system. This required machine abstraction under the SelSus ICPS federation and addition of external temperature and pressure sensors to come up with a careful setting of parameters and a BN model to recognise deterioration. This will save repair time and rejected units, and give maintenance teams an early warning and the opportunity to plan their repair during a downtime period. The benefits will be significant. Ford already installs around 4 of these machines on each assembly line it builds, and their use is still expanding due to the flexibility they bring. The principles once proven can then be applied to other critical operations.

A. RTV Machine Case

From the start to the finish the flow of RTV material though the machine experiences several failures. The dispenser (Fig. 2) is the main part of the system, it controls the exact quantity of material that need to be applied to each individual engine part. Through the cycles the RTV material can be spoiled due to air leakages combined with certain ranges of temperature and humidity which can originate clogs in the system. Fig. 2 shows complete application cycle that is repeated through the process. To eliminate and prevent issues

the machine can perform a purge operation to completely drain the system from material and this way remove any clogs.

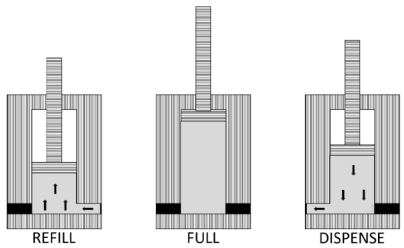

Fig. 2. RTV Material Application Cycle.

The system setup is captured by the diagram of Fig. 3. The RTV material is contained in a bucket, a pressure plate driven by a Linear Actuator (M) is put inside the bucket. The pressure applied to the plate is transmitted to the RTV which escapes though the path of least resistance, a hole in the plate connected to a flexible hose. In this section three sensors were used: 1) an Infrared (IR) wireless sensor pointing towards the bucket to assess the RTV storage temperature; 2) a Relative Humidity (RH) wireless sensor to assess the humidity condition influencing the material; 3) a cabled RAM Pressure sensor directly attached to the hose between the plate and the dispenser inlet valve. The next section is constituted by the RTV dispenser mechanism, to which two cabled sensors were directly attached: 1) a Material Feed Pressure sensor to measure the force exerted by the piston inside the dispenser; 2) a Material Feed Temperature sensor to measure the material temperature inside the chamber. Finally the hose goes trough a robotic arm that applies the bead in the engine part. To the final part of the hose two cabled sensors were physically attached: 1) a Material Application Pressure sensor, to measure the pressure between the dispenser and the application tip; 2) a Material Application Temperature sensor to measure the temperature of the RTV in the last section of the hose. The application part is contained in a closed cell where two wireless sensors are present: 1) an Application RH sensor to measure the humidity influence inside the cell; 2) an Application Ambient Temperature to measure the influence of temperature inside the cell.

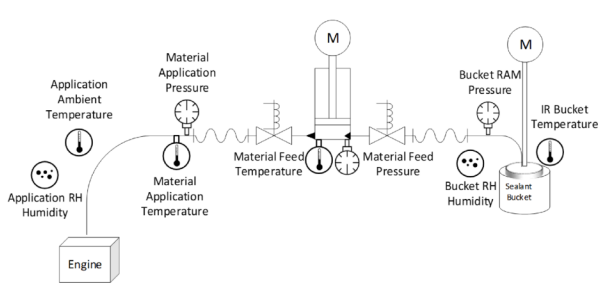

Fig. 3. RTV Machine Monitoring Setup.

Additionally to the set of sensors shown in Fig. 3, there were some additional sensors to monitor the total current consumption of the machine and of the dispenser linear actuator. These were not considered because during the three month period of sampling there was no significant variation of consumption. That led us to conclude that the increase in pressure due to clogs was not enough to disturb the current consumption profiles of both sensors and to rely mostly on the pressure sensors.

B. Bayesian Network Component

Bayesian networks have been applied in industry to successfully solve a range of different problems including troubleshooting [18], [19] as well as condition monitoring and root-cause analysis [20], [21]. The use of Bayesian networks enables machine fault detection at all levels as Component Level models can be encapsulated into Station and Factory Level models resulting in production factory wide models as considered, for instance, by [22] for root-cause analysis, equipment degradation detection as well as predictive maintenance. Knowledge driven models naturally requires access to detailed information about the system being modelled whereas data driven creation of root-cause analysis models require access to a representative set of observational data.

The overall aim of the Bayesian network component is to recognise deterioration of quality of the sealant bead produced by the RTV machine using a data driven approach where only sensor readings from the system are exploited. For this purpose operational data from the RTV machine has been collected. The challenge often faced when analysing data collected from existing industrial cyber physical systems is that significant efforts have been invested in making the systems smart, reliable, effective and self-healing. This means that systems running in a production environment may rarely malfunction or the root cause of the malfunctioning may not be easily detected at the lowest component level. Also, often the collected data consists of sensor readings, operator observations and feedback are rarely included nor is the data annotated with information on root causes in case of error or information on the settings of the machine. Therefore, we take a different approach applying a purely data-driven method with the aim of detecting issues earlier within the material delivery system by only monitoring a number of sensor readings.

Formally, a Bayesian network $N = (G, \mathcal{P})$ is a probabilistic graphical model that encodes a joint probability distribution $P(\mathcal{X})$ over a set of random variables $\mathcal{X}$ according to a DAG G such that

$$P(\mathcal{X}) \quad = \quad \prod_{P \in \mathcal{P}} P(X \mid \pi(X)), \tag{1}$$

where $\pi(X)$ are the parents of X in G [23]. The Bayesian network supports the calculation of $P(X \mid \epsilon)$ where $\epsilon = \{\epsilon_1, \ldots, \epsilon_n\}$ is a set of n variable instantiations (findings) $X = x$.

Taking the approach of [24], we aim to create a model over the sensor readings using Bayesian network learning algorithms. A greedy search-and-score (GSS) structure learning algorithm was used to induce a DAG over the sensor readings followed by a maximum likelihood estimate from

the set of complete historical data. This model cannot in itself be used for root-cause analysis or predictive maintenance. Instead we use the concept of evidence conflict to indicate a malfunctioning of the machine. The fundamental idea is that the concept of conflict of evidence can be used to identify suspicious findings pointing to problems in the sealant bead quality. The data conflict measure $conf(\epsilon)$ considered here is due to [25] and defined as

$$conf(\epsilon) = \log \frac{\prod_i P(\epsilon_i)}{P(\epsilon)}. \tag{2}$$

The assumption behind this measure is that individual findings in the evidence ϵ are positively correlated under the model. If this is not the case, then this may be caused by a data conflict producing a positive $conf(\epsilon)$.

Let $\epsilon' = \epsilon \cup \{\epsilon_{s_i}\}$ where s_i is a sensor reading, then the conflict measure decomposes as $conf(\epsilon') = \log(P(s_i)/P(s_i \mid \epsilon)) + conf(\epsilon)$ [24]. This decomposition may be exploited to monitor the readings of an individual sensor s_i through its normalised likelihood $P(s_i \mid \epsilon)/P(s_i)$, i.e., the sensor reading with the lowest normalized likelihood given the other readings has the highest contribution to the conflict. The logarithm of the inverse normalized likelihood reflects the contribution of sensor s_i to $conf(\epsilon)$ and as such it can, for instance, be used to identify sensor readings indicating a malfunctioning part of the component. For $n = 34$, we have $conf(\epsilon) = 8.78$ with $\log(P(s_i)/P(s_i \mid \epsilon \setminus \epsilon_{s_i})) = 3.67$ for material feed pressure and $\log(P(s_i)/P(s_i \mid \epsilon \setminus \epsilon_{s_i})) = 2.94$ for material application pressure. This suggests that the purge operation produces feed and application pressures that are misaligned with the other sensor readings.

C. Results

The data was collected from a RTV machine over a period of three months from late April 2017 to late July 2017. Data is collected at a frequency of one second even when the machine is idle and the data is complete, i.e., no missing values. The data contains no information on when the machine is active, which engine type is being processed nor if any failures have occurred. If the machine is idle, then the same value is repeated in the data.

Figure 4 shows the structure of the model constructed by from the collected operational data using a greedy search-and-score algorithm. The structure specifies the dependence of operational data. The data sequence (collected October, 2017) shown in Figure 5 contains three cycles with time step $n = 0, \ldots, 147$. This first cycle is a purge operation to the system to get rid of clogs after the machine has been idle for some time. The purge operation completely evacuates the material from the application container, this causes a major drop in feed pressure and sensor readings are out of sync. The purge operation is followed by two cycles of the machine where the sealant bead is applied to engine blocks. Figure 6 shows the running average of the data conflict measure for the operational produced by the RTV machine. To make the curve smooth and make the measure less sensitive to fluctuating values in the sensors, a running average over five seconds is applied. It is clear that the model detects the unexpected behaviour of the sensor readings during the purge operation and the two following operation cycles do not produce any indication of issues with the operation of the machine and thereby the quality quality of the bead.

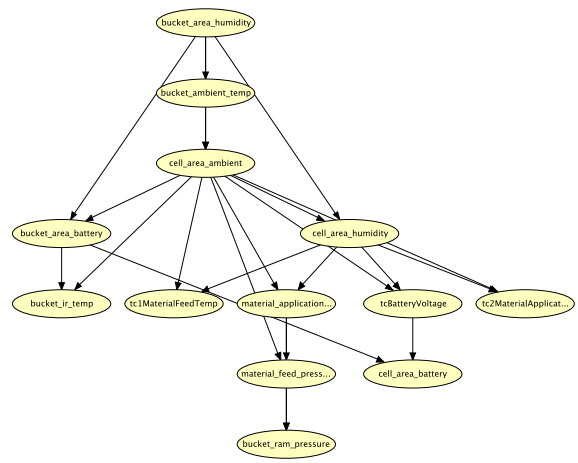

Fig. 4. The structure of the model constructed from operational data.

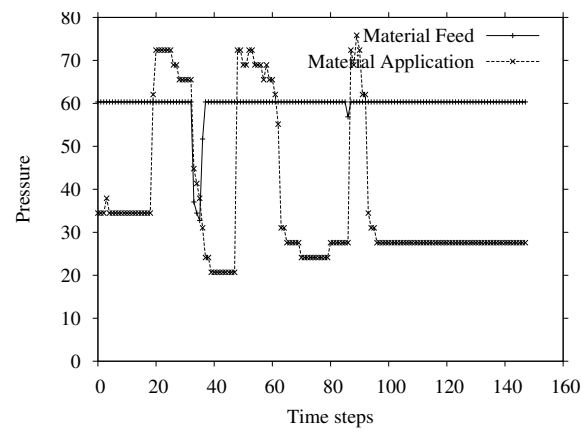

Fig. 5. Sequence of sensor readings.

Let s_i be a sensor with reading ϵ_{s_i}. The contribution of s_i to $conf(\epsilon)$ is determined by the normalised likelihood $P(\epsilon_{s_i})/P(\epsilon_{s_i} \mid \epsilon \setminus \epsilon_{s_i})$. Hence, we can use this value to identify the main contributing factors to the data conflict. This can be used in a diagnostics process even when root causes are not represented in the model. At time step $n = 34$, $conf(\epsilon) = 8.68$ with $\log P(\epsilon_{s_i})/P(\epsilon_{s_i} \mid \epsilon \setminus \epsilon_{s_i}) = 3.67$ for s_i equal to material feed pressure and $\log P(\epsilon_{s_i})/P(\epsilon_{s_i} \mid \epsilon \setminus \epsilon_{s_j}) = 2.94$ for s_j equal to material application pressure. This suggests that these measurements are not aligned with the other sensor readings, which can be explained by the purge operation.

V. Conclusions

The data conflict measure is based on comparing the probability of the evidence between two different models. In particular, the marginal independence model is used as the

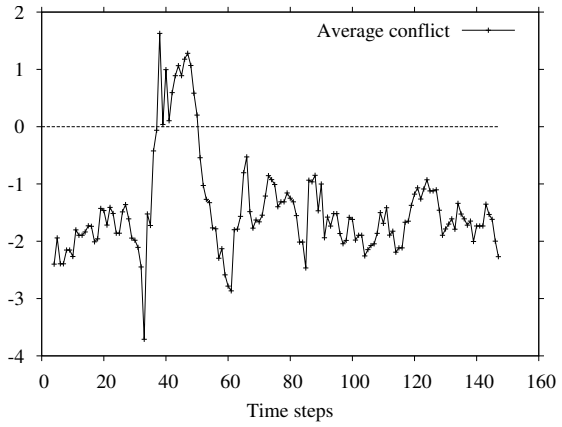

Fig. 6. Plot of the running average data conflict measure for a single operational cycle.

straw man model. More advanced models can be considered. Also, the decomposition of data conflict measure into local conflicts makes it possible to trace the origin of a, e.g., Station Level conflict to a Component Local conflict. The self-contained model of the equipment at the Component level can be integrated into wider Station or Factory Level models through the use of object-oriented modelling.

Parameter learning algorithms for continuous model improvement using operational data [26] are being considered. However, here there is a challenge that the parameters of the model may adapt to a malfunctioning sensor reading.

The Sensor SelComp can extend its compliance with other ICPS's simply by adopting standardized data representation schemes such as AutomationML [27]. The issue of incompatibility between algorithms implemented in different languages is a priority for applicability to real world. A final consideration is real-time capability, which was not a requirement in this work, but constitutes an interesting capability if combined with fault detection modules for control applications.

Acknowledgments

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.

References

[1] S. Yin and O. Kaynak, "Big data for modern industry: challenges and trends [point of view]," *Proceedings of the IEEE*, vol. 103, no. 2, pp. 143–146, 2015.

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

[3] K.-K. Lau and Z. Wang, "Software component models," *IEEE Transactions on software engineering*, vol. 33, no. 10, pp. 709–724, 2007.

[4] C. Maga, N. Jazdi, and P. Göhner, "Reusable models in industrial automation: experiences in defining appropriate levels of granularity," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 9145–9150, 2011.

[5] F. Fouquet, B. Morin, F. Fleurey, O. Barais, N. Plouzeau, and J.-M. Jezequel, "A dynamic component model for cyber physical systems," in *Proceedings of the 15th ACM SIGSOFT symposium on Component Based Software Engineering*. ACM, 2012, pp. 135–144.

[6] A. Beugnard, J.-M. Jézéquel, N. Plouzeau, and D. Watkins, "Making components contract aware," *Computer*, vol. 32, no. 7, pp. 38–45, 1999.

[7] S. Balaji and M. S. Murugaiyan, "Waterfall vs. v-model vs. agile: A comparative study on sdlc," *International Journal of Information Technology and Business Management*, vol. 2, no. 1, pp. 26–30, 2012.

[8] L. Neto, J. Reis, R. Silva, and G. Gonçalves, "Sensor selcomp, a smart component for the industrial sensor cloud of the future," in *Industrial Technology (ICIT), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1256–1261.

[9] 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), 2014 IEEE*. IEEE, 2014, pp. 1–8.

[10] L. Wang and A. Haghighi, "Combined strength of holons, agents and function blocks in cyber-physical systems," *Journal of Manufacturing Systems*, vol. 40, pp. 25–34, 2016.

[11] Apache Software Foundation, "Maven 3," https://maven.apache.org, 2017, accessed: 2017-12-12.

[12] Mozzila, "Rhino javascript compiler," https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino/JavaScript_Compiler, 2007, accessed: 2017-12-12.

[13] K.-K. Lau and Z. Wang, "A taxonomy of software component models," in *31st EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, 2005, pp. 88–95.

[14] I. Crnkovic, S. Sentilles, A. Vulgarakis, and M. R. Chaudron, "A classification framework for software component models," *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 593–615, 2011.

[15] P. Hošek, T. Pop, T. Bureš, P. Hnětynka, and M. Malohlava, "Comparison of component frameworks for real-time embedded systems," in *International Symposium on Component-Based Software Engineering*. Springer, 2010, pp. 21–36.

[16] G. Doukas and K. Thramboulidis, "A real-time-linux-based framework for model-driven engineering in control and automation," *IEEE Transactions on Industrial Electronics*, vol. 58, no. 3, pp. 914–924, 2011.

[17] P. Adolphs, H. Bedenbender, D. Dirzus, M. Ehlich, U. Epple, M. Hankel, R. Heidel, M. Hoffmeister, H. Huhle, B. Kärcher *et al.*, "Status report-reference architecture model industrie 4.0 (rami4. 0)," *VDI-Verein Deutscher Ingenieure eV and ZVEI-German Electrical and Electronic Manufacturers Association, Tech. Rep*, 2015.

[18] J. Breese and D. Heckerman, "Decision-theoretic troubleshooting: A framework for repair and experiment," in *UAI*, March 1996, pp. 124–132.

[19] M. S. Sayed and N. Lohse, "Printer troubleshooting using bayesian networks," *Intelligent Problem Solving. Methodologies and Approaches*, pp. 367 – 379, 2000.

[20] G. Weidl, A. L. Madsen, and E. Dahlquist, "Decision support on complex industrial process operation," in *Bayesian Networks: A Practical Guide to Applications*. Wiley, 2008, ch. 18, pp. 313–328.

[21] G. Weidl, A. L. Madsen, and S. Israelson, "Applications of object-oriented Bayesian networks for condition monitoring, root cause analysis and decision support on operation of complex continuous processes," *Computers and Chemical Engineering*, vol. 29, pp. 1996–2009, 2005.

[22] A. L. Madsen, N. Søndberg-Jeppesen, N. Lohse, and M. S. Sayed, "A Methodology for Developing Local Smart Diagnostic Models Using Expert Knowledge," in *Proceedings of the 2015 IEEE International Conference on Industrial Informatics (INDIN)*, 2015, p. 6.

[23] U. B. Kjærulff and A. L. Madsen, *Bayesian Networks and Influence Diagrams: A Guide to Construction and Analysis*, 2nd ed. Springer, 2013.

[24] T. Nielsen and F. Jensen, "Alert systems for production plants: A methodology based on conflict analysis," in *ECSQARU*, 2005, pp. 76–87.

[25] F. Jensen, B. Chamberlain, T. Nordahl, and F. Jensen, "Analysis in HUGIN of data conflict," in *UAI*, 1991, pp. 519–528.

[26] A. Madsen, N. Sondberg-Jeppesen, F. Jensen, M. Sayed, U. Moser, L. Neto, J. Reis, and N. Lohse, "Parameter learning algorithms for continuous model improvement using operational data," in *ECSQARU*, 2017, pp. 115–124.

[27] R. Drath, A. Luder, J. Peschke, and L. Hundt, "Automationml-the glue for seamless automation engineering," in *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2008.* IEEE, 2008, pp. 616–623.