Performance assessment of the integration between industrial agents and low-level automation functions

Luis Ribeiro*, Stamatis Karnouskos†, Paulo Leitão‡, José Barbosa‡, Martin Hochwallner*

*Linköping University, SE-58 183 Linköping, Sweden, email: {luis.ribeiro, martin.hochwallner}@liu.se
†SAP, Walldorf, Germany, email: stamatis.karnouskos@sap.com
‡Research Centre in Digitalization and Intelligent Robotics (CeDRI), Instituto Politécnico de Bragança, Campus de Santa Apolónia, 5300-253 Bragança, Portugal, email: {pleitao, jbarbosa}@ipb.pt

***Abstract*—The increasing need for more adaptive production environments is a big motivator for the adoption of agent-based technologies in industrial systems, as they provide better mechanisms for handling dynamically and intelligently various kinds of production disturbances. Unlike with the utilization of most conventional automation languages, the use of agents enables, in an easy way, the setup of dynamic and autonomous adaptive processes to handle large and complex engineering system functions and interactions. Agent-technologies in cyber-physical systems contexts require at some point integration with automation controllers. However, most commonly available and used agent system implementations in the industry were not designed for hard real-time control use cases, and do not utilize real-time operating systems or dedicated hardware. Hence, they cannot match the hard-real-time performance of automation controllers. This work provides some insights on the performance that can be achieved with agent-based approaches that integrate with low-level automation system functions. It considers the performance of the agent-based practices in light of non-real-time dedicated hardware or operating systems. The results show that agents are well suited for the majority of soft-real-time control applications.**

I. Introduction

The concept of a smart factory, where intelligent resources interact with each other and continuously adjust their behavior to changes, opportunities, and disturbances, is increasingly seen as the answer to attain more sustainable production practices at a time where product customization and complexity are on the rise [1], [2].

Agent technologies have, for a long time, played an essential role in influencing the design of the smart factory by promoting distribution, decentralization, intelligence, autonomy, and adaptation, contributing to achieving flexibility, robustness, responsiveness, and reconfigurability [3], [4]. Nevertheless, most of the experiments with such technologies and principles have been restricted to smaller scale industrial demonstrators and less commonly introduced in production. Among the identified key factors that contribute to the adoption of agents in industrial domain [5], are the technology (which includes legacy system integration) and standardization issues. Their scope of applicability and design rules have also been frequently misunderstood which has not facilitated their adoption [6].

The consolidation of some IT (Information Technologies) / AI (Artificial Intelligence) technologies and the pervasiveness of computer networks has re-ignited both the interest and opened for new opportunities in multi-agent-based control. This new wave of intelligent systems is going under the brand of Cyber-Physical Systems (CPS). CPS stand for most of what has been demonstrated with agents before but emerges at the time where both concepts and technologies are starting to be better understood. They are also transversal, cutting across several domains: energy, health, agriculture, transportation, social networking, surveillance, and defense, etc. [1], [7].

Current agent-technologies are well-suited to manage complex and large engineering ecosystems such as a smart factory. Previous work in multi-agent systems applied to production automation has indicated a superior performance in managing uncertainties and catering for unexpected events [8] but has also exposed the deficiencies in handling hard-real-time events. For that reason, researchers and practitioners have traditionally used agents in combination with conventional automation controllers. This enables a proper separation of concerns regarding the action scope and objectives of the different concepts and technologies.

Several integration and interaction practices, connecting these two worlds, have emerged independently and been dispersed in the literature. Until recently, very little work has been done to characterize the adequacy of the practices in different contexts [9]. Existing standards for software products such as ISO/IEC 25010 could be relevant for assessing the practices [10].

When considering the utilization of agents in industrial settings, one key quality sought is related to the performance of the agent solution. However, due to lack of standardized interfaces and detailed performance measurements, it is difficult to compare industrial agent practices, particularly using different interface approaches to interconnect software agents and physical automation devices. In this context two major Research Questions (RQ) arise:

- RQ1: Is the performance of the interface practice adequate for industrial system control? What levels of performance can be expected?
- RQ2: Is the industrial agent-based solution behavior stable, and if not, which factors affect it?

To approach these research questions, several aspects related to the performance need to be investigated. For instance, RQ1

requires an investigation of different integration patterns and absolute statistical values pertaining to the performance e.g. by measuring the round trip time (RTT) of the agent and device interaction. For RQ2 a statistical analysis of thousands of requests needs to be made. The aim of this work is to try to empirically approach these two RQs along the line of thought discussed.

The rest of the paper is organized as follows: section II briefly discusses the related work that supports the present research, and section III presents the reasons for the methodological approach and the selected test cases; section IV focuses on the empirical results achieved; section V discusses the implications of the findings, especially in the context of the set of defined research questions; and finally, in section VI the conclusions and future research and development directions are presented.

II. Related Work

The connection between agent-based systems and conventional automation controllers has been explored for the past ten years and different practices have emerged. The typical practice realizes a two-layered system, similar to the holon and mechatronic concepts, comprising: (i) a low level control layer (LLC), consisting of system specific high performance controller code, and (ii) a high level control layer (HLC) dominated by agents or higher order control entities incapable of delivering real-time performance but able to provide intelligence to decision-making.

Early works seeking a more standardized integration of the HLC and the LLC layers include [11]–[14]. In these approaches, the dominant technologies on the LLC layer include code developed by using programming languages from IEC 61131-3 [15] and IEC 61499 standards, while the HLC layer has been predominantly dominated by agent-based implementations. The implementation of the two-layered interface system based on the IEC 61131-3 standard for the LLC layer has resorted to distinct connectivity solutions, but the ones based on IEC 61499 have used mechanisms that have been considered in the standard for this specific purpose.

A more recent study [9] has uncovered a wider range of different integration practices and patterns across three application domains: factory automation, power and energy systems and building automation. The surveyed practices confirm the prevalence of already mentioned technologies but exposed a larger set of interaction and communication protocols as well as HLC integration practices. Specifically, the HLC can be deployed within and outside the automation controller and its interaction with the LLC ranges from almost direct control through shared memory space to modern brokered message-based interaction.

Of particular relevance is the conclusion that according to the location of the HLC (e.g., the software agent) and the LLC (e.g., the low-level automation function), the HLC and the LLC can share the same computational platform (on-device) or can be in different ones (i.e. hybrid). Depending on the scope, the HLC can exert more direct control on the LLC (tightly-coupled) or the interaction can be mediated by a broker (loosely-coupled).

In this work, the focus is on the tightly coupled design, where the HLC has a permanent, non-mediated, connection to the LLC. Such connection can exist in the form of direct network communication or shared memory. In this abstraction level, three designs are possible. The first one consists of computationally separating the HLC and LLC, as represented in Figure 1, with HLC accessing remotely to the LLC. Both ends will implement communication mechanisms that enable them to interact. This is affected, by the design, the quality of the network infrastructure and the computational capabilities at both ends.

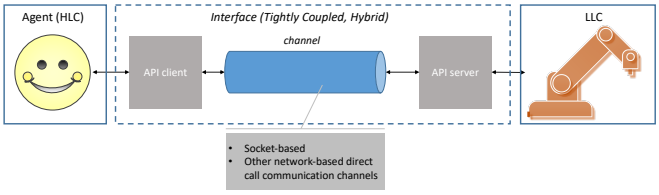

Figure 1. Structure for the Hybrid Tightly Coupled interface

Embedding the HLC into the low-level control device, as illustrated in Figure 2, mitigates the network related issues and should improve the overall performance if the device has enough computational power to support both the HLC and the LLC while fulfilling the timing requirements of both.

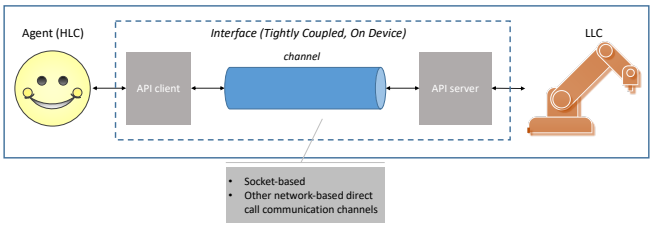

Figure 2. Structure for the On Device Tightly Coupled interface

In the very extreme case, the HLC may be compiled together with the LLC creating a fully coupled structure without any significant computational separation. This version of the design attempts to ensure maximum performance, but as a drawback, it may limit the reconfiguration capabilities of the agent.

III. Testing Methodology

To answer the posed research questions, a structured test plan has been derived. For RQ1, in order to investigate the performance, absolute values are needed. As such, it was decided to investigate three different parameters:

(i) Inter-Tick Time (ITT) – measures the agent's cycle time (i.e., the time between executions);

(ii) Tick Time (TT) – measures the agent's total execution time per cycle (i.e., from the moment it starts executing the behavior that includes the call to the controller to the moment it finishes it);

Preprint version – Accepted for publication at IEEE INDIN 2018

(iii) Round Trip Time (RTT) – is measured from the moment the agent issues a command to the controller to the point where it receives a reply for that command from the controller it is communicating with.

Simply put, ITT contains TT which contains RTT and, collectively, they enable measuring different execution moments in the interaction between the HLC and the LLC as depicted in Figure 3.

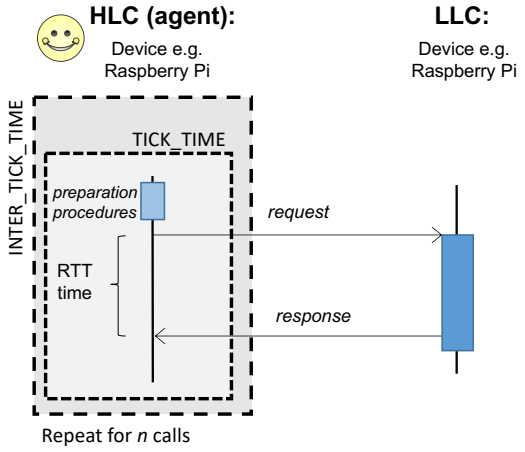

Figure 3. Probing points in the testing agent lifecycle

TT, ITT, and RTT are known to be influenced by the underlying implementation and the computational platform supporting it. However, TT and ITT are also strongly influenced by the selection of the agent platform and the additional application-specific code. Considering that the present paper is focusing on the interaction between HLC and LLC, it places a considerable focus on the RTT analysis. TT and ITT provide insight into the operation of the HLC itself and are, in the present context, only generally relevant to understand the quality of the tested implementations and their suitability for hard-real-time operation since the focus of the work lies on the interaction between HLC and LLC.

For RQ2, in order to find out if the behavior is stable, tests need to be carried out that stress the system beyond its normal operations. Here we consider that agents are not hard-real-time entities and that they operate in an event-driven time-independent manner. With the currently available technology, they execute (HLC-level) is the hundreds of milliseconds scale. At this scale, for instance, issuing a large number of requests (e.g., 200000) as quickly as possible, should reveal if the performance is stable over all these consecutive tests, but at the same time should not happen frequently in real operation at HLC level for most automation processes. To approach the second part of this RQ and find out the potential impact of individual factors, one needs to vary them individually, for example the number of requests from the agent to the device, the message size (payload) and experiments with communication patterns (e.g., as fast as possible or in defined cycles).

Table I
OVERVIEW OF EXECUTED TESTS AND PARAMETER SCALING

Test	Number of Calls	Message Size (Bytes)	Target Cycle Time (ms)
1	10-10-100	1000	10
2	100-100-1000	1000	10
3	1000-1000-50000	1000	10
4	10-10-100	1000	freewheeling
5	100-100-1000	1000	freewheeling
6	1000-1000-50000	1000	freewheeling
7	50000-5000-200000	1000	freewheeling
8	10-10-100	2000	10
9	100-100-1000	2000	10
10	1000-1000-50000	2000	10
11	10-10-100	2000	freewheeling
12	100-100-1000	2000	freewheeling
13	1000-1000-50000	2000	freewheeling
14	50000-5000-200000	2000	freewheeling
15	1000	100-100-1000	freewheeling
16	1000	100-100-1000	10
17	1000	1000-1000-10000	freewheeling
18	1000	1000-1000-10000	10
19	1000	1000	1-1-20
20	1000	1000	10-10-100

In order to approach both RQs empirically, as discussed, a set of 20 classes of tests were defined for the different physical testing setups. There are three main identified variables:

(i) number of calls from the HLC to the LLC;
(ii) target cycle time of the agent;
(iii) the size of the message being transmitted.

A summary of the 20 classes of tests, detailed in Table I, was therefore executed. The pattern A-B-C denotes scaling of the specific test parameter from A to C in increments of B. The message size is kept moderate, above and below to the IP fragmentation & reassembly level that is defined by the maximum packet length limit of 1500 Bytes. The target cycle time is set in ms, or as "freewheeling" which indicates "as fast as possible" behavior.

Table II
HYBRID TIGHTLY COUPLED CASE SUMMARY

Case	HLC	LLC	Connection Type & Behaviour
1	Raspberry Pi 1 Model B+	Raspberry Pi 1 Model B+ with embrick Z-RaspberryBrick-01 (expansion board) and embrick G-8Di8Do-01 (expansion board) [16]	The LLC runs an socket server that accepts commands from the agent and executes them. The execution involves reading the message payload, and writing and reading a random 16-bit value to and from its digital I/Os before sending back a copy of the initial message to the agent. The communication is done over a 100 Mbps Ethernet.
2	Raspberry Pi 1 Model B+	Raspberry Pi 1 Model B+	Similar to case 1 but without the hardware write and read operations.

The experimental setup to test the selected integration cases needs to fulfill a number of conditions in order to allow the replication of the experiments. The test focuses on the integration itself and the conditions on the agent, and the controller sides need to be kept controlled. With this rationale in mind, it was decided that the agent component (HLC) in the hybrid model would be executed on several platforms shown

Preprint version – Accepted for publication at IEEE INDIN 2018

in Table II. The on-device cases are executed similarly on the selected platforms shown in Table III.

Table III
ON DEVICE TIGHTLY COUPLED CASE SUMMARY

Case	HLC & LLC	Connection Type & Behaviour
1	Raspberry Pi 1 Model B+ with embrick Z-RaspberryBrick-01 (expansion board) and embrick G-8Di8Do-01 (expansion board) [16]	Same as case 1 in Table II with the corresponding adaptation for the On Device case
2	Raspberry Pi 3	local call
3	MacOS 10.13 (Intel Core i7, 2.3 GHz, 16 GB RAM)	local call
4	Windows 10 (Intel Core i5, 2.3 GHz, 8GB RAM)	local call

The agent software is implemented in JADE version 4.4 [17]. JADE is one of the most widely-used platforms for developing agent-based systems; it is FIPA (Foundation for Intelligent Physical Agents) compliant and thoroughly documented. Agents within an agent-based system often handle more than one task simultaneously. It is impossible to predict the load of a specific agent as this is always application specific. For this reason, the implementation of the testing agent considers that each agent is running one behavior only, cyclically, with a reconfigurable target cycle time. The different Raspberry Pi platforms used the same image of the operating system, i.e., 32-bit Raspbian GNU/Linux 8.10, and Java 8.

IV. RESULTS

The result of performing tests 1–20 on the 6 test cases described in Table II and Table III has provided several relevant insights to the pursued RQs as well as about the differences among the selected cases.

One significant differentiator is whether the tests are carried out in freewheeling mode or on fixed-cycle, e.g., 10 ms. Overall, the freewheeling execution mode was consistently faster across the entire battery of tests. This could be even seen in the tests with a $1\,\mathrm{ms}$ fixed cycle time. However, at this pace, the agent is generally not able to keep-up with the cycle time. The measurements done in both cases are in general similar, with the corresponding offset for the fixed cycle cases. However, the freewheeling tests will generally consume a significant portion of the available computing time in comparison with the fixed period tests. Concerning variations within a fixed cycle time the observations suggest that the HLC implementation struggles to keep up with short cycle times ($[1\,\mathrm{ms},3\,\mathrm{ms}]$) but generally fulfills higher cycle times.

In the tests with varying payload, the results are too enough for any significant difference to be noticeable and practically relevant. This is true also for the scenarios where the HLC and LLC were in the same physical machine and for the remote tests. We have not carried out tests with large sizes, as these do not correspond to typical cases in industrial automation, where it is more often, that many messages, typically small in size are exchanged (rather than few but big in size messages).

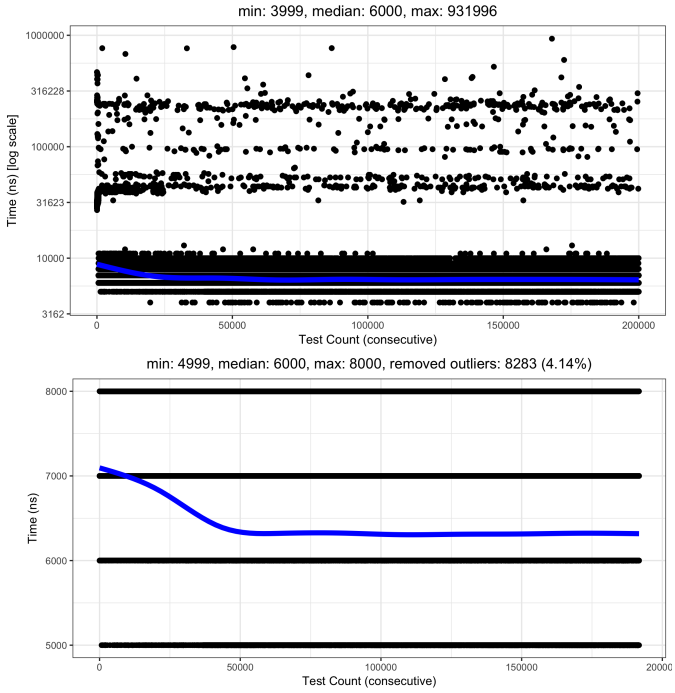

Figure 4. RTT for 200000 requests from Raspberry Pi to embrick with full data (top) and with outliers removed (bottom).

Finally, the number of calls seems to affect the performance of the instantiated cases. In the tests for cases 1–15, it was possible to observe an important increase in the performance between cycles 1000 and 2000 (as shown in Figure 5 and Figure 4). This was consistent across all tests and had an impact on all the measurements (RTT, TT, and ITT). For example the RTT values consistently dropped from approximately $[25\,\mu\mathrm{s},40\,\mu\mathrm{s}]$ to approximately $[8\,\mu\mathrm{s},11\,\mu\mathrm{s}]$ in case 1 (Table II). The sudden increase in performance shown at Figure 5 (top), was attributed to Java Virtual Machine (JVM) optimizations at runtime. By explicitly switching off the optimization, the results in Figure 5 (bottom) were attained, where it is clear that the initial performance gap was smoothed out. However, as seen, although the performance is stable, it is consistently worse.

All of the tests show some outliers, which were hypothesized since we do not utilize a real-time platform. Removing the outliers enables us to obtain more detailed insights on the actual micro-behaviors. For instance, Figure 4 shows on top the view with the outliers, which if removed, reveal the bottom figure that provides insights into the actual measurements carried out. However, outliers cannot generally be ignored. Figure 4 (top) shows an important distribution of outliers (roughly 4.14%) approximately in the range $[30\,\mu\mathrm{s},310\,\mu\mathrm{s}]$ with a small percentage approaching $1\,\mathrm{ms}$. If one concentrates on the central part of the distribution (bottom Figure 4) then the majority of the samples lie in the interval $[5\,\mu\mathrm{s},8\,\mu\mathrm{s}]$. The graph shows the data points compressed into several bands. Such behavior is a direct consequence of the resolution of Java's nanosecond time measuring function on the test

Preprint version – Accepted for publication at IEEE INDIN 2018

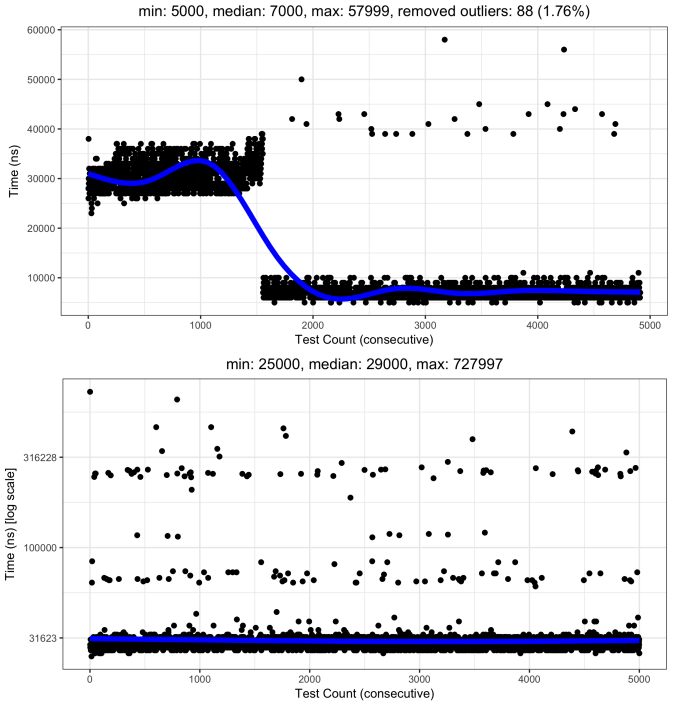

Figure 5. RTT for 5000 requests from Raspberry Pi to embrick with Java optimization on (top figure) and turned off (bottom figure).

devices. The only guarantees provided by this timer are that the precision is in nanoseconds and the resolution, being system dependent, is at least milliseconds.

Finally, as expected, the performance of the tests is qualitatively proportional to the computational power available both in the HLC and LLC platforms. This is evidenced in multi-core platforms (e.g., the Raspberry Pi 3) where the overall performance is considerably better. In addition, depending on the GC strategy, the presence of more memory as well as the parallel usage of many CPU cores, minimizes the GC interference in the performance.

V. Discussion

The RQ1 posed the question of performance suitability for industrial scenarios and the performance levels that can be attained. The results in section IV show examples of the time behavior in the investigated practices. They clearly depict a "best effort" performance from the HLC, in keeping up with potential LLC performance requirements but also with a significant outliers population. This suggests that the practices are not compatible with hard-real-time requirements of many industrial cases, especially with those under the 1 ms threshold.

JADE is considered good enough for achieving soft-real-time constraints; however hard real-time aspects cannot be guaranteed [18]. Although low execution times can be reached, that make software agents in JADE appropriate for a wide range of control scenarios, closed-loop controls and other critical missions that rely on low execution time variations need further work in the direction of a fully-fledged real-time OS and appropriate utilization at the agent level.

RQ2 posed the question of behavior stability and factors that affect it. As shown in section IV, there is behavior variability and some factors were identified. Some of the effects measured and discussed are common in the Java world. In particular, the used agent platform (JADE) is implemented in Java. Java applications are known to include optimizations, which may vary the behavior over time [19], [20]. In addition, they may slow down when the Java Garbage Collector (GC) becomes active. Even if in all the tests specified in Table I the heap size of the JVM has been set to 256 MB, garbage collection is almost inevitable for the most intensive tests. It is known that Java's GC is a major source of performance variability in a real-time context [20], [21]. By scaling the number of messages and controlling the target cycle time of the agent, the rate of object creation and destruction is necessarily varying.

While Java optimization was clearly evident by the improvement of the RTT performance, GC impacted the performance, as attested by the outlier population. Such variability in behavior has also puzzled others, especially in more complex settings with multiple agents, where the interactions among them make it more challenging to investigate it [22]. Overall for small execution bursts (tests 1, 2, 4, 5, 8, 9, 11 and 12 in Table I) the GC had, generally, a minimum interference. For longer execution bursts (tests 3, 6, 7, 10, 13 and 14 in Table I) its action was noticeable.

The performance of the tested scenarios is somehow more stable if Java optimization is turned off. However doing so entails a relevant performance penalty. An interesting option may be controlling this optimization procedure by shortening the number of executions that Java should use to base its optimization on. In these cases, a middle-ground between behavior stability and performance was attained. Nevertheless, and as expected, hard real-time performance could never be attained during the tests and this arises from the nature of the programming tools, platforms and communication protocols selected. For industrial scenarios, the exact use-case requirements need to be considered, including the number of calls to be made by the agent as well as the interval, in order to judge if such performance effects do make a difference in practice.

This work adopts a rather simplistic view that in an hard-real-time system all the deadlines must be strictly met, while in a soft-real-time some deadlines may be missed which will result in acceptable performance degradation. In this context, considering that existing agent-based systems cannot meet hard real-time requirements, the present tests indicate that, in applications requiring hard-real-time assurances, the LLC should provide these guarantees while ensuring that its performance gracefully degrades due to HLC delays. The HLC appears to have a stable response over time with its performance degrading periodically due to the action of Java's GC but without any important cumulative effect over time. This is encouraging as it enables the design of the LLC in a compatible way with these periodic perturbations.

Finally, the relatively simple setup used did not produce

Preprint version – Accepted for publication at IEEE INDIN 2018

meaningful differences between separating the HLC and LLC in terms of performance (cases 1 in both Table II and Table III). Such results though should be taken with a grain of salt, since both the HLC and LLC implementations in this work have been reduced to an absolute minimum. It may be reasonable to think that a typical HLC and LLC full-featured implementation would have a considerably higher impact on computational resources. In this case, the CPU and available memory may become bottlenecks, especially considering the typical cases of embedded devices.

VI. Conclusions

Two typical interaction patterns for the integration between Industrial Agents and a low-level automation function have been empirically assessed. On both patterns, the agent exerts direct control by communicating with the low-level controller, which acts as a command interpreter. The results confirm literature findings and re-enforce the notion that the set of wide-spread supporting agent technologies cannot operate in hard-real-time environments, without explicitly focusing on real-time stacks. However, as discussed, there is a wide range of soft-real-time scenarios where direct control would be feasible.

In the soft-real-time context, as evidenced by the empirical approach in this work, some considerations emerge. Even if the RTT measurements are consistently within the µs region, the HLC implementation reduces the scope of actuation to the ms region on the tested scenarios. Performance can be enhanced by increasing the computational capabilities of the supporting physical platforms. Here CPU performance and availability have an important impact.

The performance is also very sensitive to the configuration of the JVM and interesting results can be attained if the agent is executing repetitive tasks by forcing the optimization to operate prematurely. Disabling the optimization completely will lead to a more predictable behavior at a performance penalty. This is generally not recommended since it does not lead to hard-real-time capable execution anyway.

The tests have not shown cumulative performance degradation in the long run. This is an important conclusion since it indicates that it is possible to integrate the HLC and LLC in a way hard-real-time is guaranteed and can address the worst case scenarios of the HLC in an acceptable way.

Overall the quantitative results show a higher range of potential applications than generally expected, considering the characteristics of the tested hardware and software. This somehow contradicts the popular idea that the role of agents should be confined to higher abstraction layers with very limited timing constraints.

References

[1] P. Leitão, S. Karnouskos, L. Ribeiro, J. Lee, T. Strasser, and A. W. Colombo, "Smart agents in industrial cyber–physical systems," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 1086–1101, May 2016.

[2] S. Karnouskos, A. W. Colombo, and T. Bangemann, "Trends and challenges for cloud-based industrial cyber-physical systems," in *Industrial Cloud-based Cyber-Physical Systems: The IMC-AESOP Approach*. Springer, May 2014, pp. 231–240.

[3] P. Leitão, "Agent-based distributed manufacturing control: A state-of-the-art survey," *Engineering Applications of Artificial Intelligence*, vol. 22, no. 7, pp. 979–991, Oct. 2009.

[4] P. Leitão and S. Karnouskos, Eds., *Industrial Agents: Emerging Applications of Software Agents in Industry*. Elsevier, Mar. 2015.

[5] S. Karnouskos and P. Leitão, "Key contributing factors to the acceptance of agents in industrial environments," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 696–703, Apr. 2017.

[6] L. Ribeiro and M. Björkman, "Transitioning from standard automation solutions to cyber-physical production systems: An assessment of critical conceptual and technical challenges," *IEEE Systems Journal*, pp. 1–13, 2017.

[7] S. K. Khaitan and J. D. McCalley, "Design techniques and applications of cyberphysical systems: A survey," *Systems Journal, IEEE*, vol. 9, no. 2, pp. 350–365, Jun. 2015.

[8] P. Valckenaers and H. Van Brussel, *Design for the unexpected: From holonic manufacturing systems towards a humane mechatronics society*. Butterworth-Heinemann, 2015.

[9] P. Leitão, S. Karnouskos, L. Ribeiro, P. Moutis, J. Barbosa, and T. I. Strasser, "Common practices for integrating industrial agents and low level automation functions," in *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*, IEEE. IEEE, Oct. 2017, pp. 6665–6670.

[10] S. Karnouskos, R. Sinha, P. Leitão, L. Ribeiro, and T. Strasser, "Assessing the integration of software agents and industrial automation systems with iso/iec 25010," in *IEEE International Conference on Industrial Informatics (INDIN)*, 2018.

[11] O. J. L. Orozco and J. L. M. Lastra, "A real-time interface for agent-based control," in *Industrial Embedded Systems, 2007. SIES'07. International Symposium on*. IEEE, 2007, pp. 49–54.

[12] I. Hegny, O. Hummer, A. Zoitl, G. Koppensteiner, and M. Merdan, "Integrating software agents and iec 61499 realtime control for reconfigurable distributed manufacturing systems," in *Industrial Embedded Systems, 2008. SIES 2008. International Symposium on*. IEEE, 2008, pp. 249–252.

[13] P. Vrba, P. Tichy, V. Marík, K. H. Hall, R. J. Staron, F. P. Maturana, and P. Kadera, "Rockwell automation's holonic and multiagent control systems compendium," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 41, no. 1, pp. 14–30, 2011.

[14] P. Leitão, A. W. Colombo, and S. Karnouskos, "Industrial automation based on Cyber-Physical Systems technologies: Prototype implementations and challenges," *Computers in Industry*, vol. 81, pp. 11–25, Sep. 2015.

[15] *IEC 61131-3:2013, Programmable controllers – Part 3: Programming languages*, International Electrotechnical Commission (IEC) Std. [Online]. Available: https://webstore.iec.ch/publication/4552

[16] "emBrick G-8Di8Do-01 module." [Online]. Available: http://www.embrick.de/products/g-8di8do/

[17] F. L. Bellifemine, G. Caire, and D. Greenwood, *Developing multi-agent systems with JADE*. Chichester, England Hoboken, NJ: John Wiley, 2007.

[18] D. Krol and F. Nowakowski, "Practical performance aspects of using real-time multi-agent platform in complex systems," in *2013 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, oct 2013.

[19] V. Horký, P. Libič, A. Steinhauser, and P. Tůma, "DOs and DON'ts of conducting performance measurements in java," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering - ICPE '15*. ACM Press, 2015.

[20] D. Gu, C. Verbrugge, and E. M. Gagnon, "Relative factors in performance analysis of java virtual machines," in *Proceedings of the 2nd international conference on Virtual execution environments - VEE '06*. ACM Press, 2006.

[21] P. Libič, P. Tůma, and L. Bulej, "Issues in performance modeling of applications with garbage collection," in *Proceedings of the 1st international workshop on Quality of service-oriented software systems - QUASOSS '09*. ACM Press, 2009.

[22] G. Polakow, "JADE environment performance evaluation for agent-based continuous process control algorithm," in *2016 21st International Conference on Methods and Models in Automation and Robotics (MMAR)*. IEEE, aug 2016.

Preprint version – Accepted for publication at IEEE INDIN 2018