# Low-Level Service Management in Cyber-Physical Systems

Borja Bordel[1,2(✉)], Ramón Alcarria[1,2], Diego Sánchez-de-Rivera[2], and Antonio Jara[1]

[1] Institute of Information Systems, University of Applied Sciences Western Switzerland, Delémont, Switzerland
ramon.alcarria@upm.es, jara@ieee.org

[2] Universidad Politécnica de Madrid, Madrid, Spain
{bbordel,diegosanchez}@dit.upm.es

**Abstract.** First standardization initiatives of the Cyber-Physical Systems (CPS) paradigm face a type of solutions with a top-down approach. In this view, user services and applications are transformed, decomposed and delegated until they are finally executed by hardware devices. However, most works do not describe the final execution phase, when a certain device is selected to perform an action. Therefore, in this paper we describe a management solution to coordinate the execution of low-level services in CPS. The solution employs a probabilistic selection technique based on the concept of Cost and Quality-of-Service, and includes both an orchestration algorithm and a choreography procedure. The proposal includes, moreover, a general framework explaining all the management levels and an experimental validation which evaluates the performance of the proposed technology.

**Keywords:** Cyber-Physical Systems · Low-level services · Service management · Hardware devices

## 1 Introduction

Cyber-Physical Systems (CPS) are integrations of physical and computational processes [1]. This general definition, however, does not allow describing a uniform architecture or implementation, so several proposals may be found [2, 3]. In this context, various authors and standard organizations are trying to fix the CPS paradigm defining different reference architectures. One of the most important it is the NIST architecture [4]. On the other hand, all these proposals follow a top-down approach. In these systems, firstly, user applications and services are defined at high-level and, later, they are transformed, decomposed and delegated in order to request hardware devices to perform the corresponding actions [5].

B. Bordel, R. Alcarria and D. Sánchez-de-Rivera—On leave from Universidad Politécnica de Madrid, Madrid, España

L. Barolli and T. Enokido (eds.), *Innovative Mobile and Internet Services in Ubiquitous Computing*, Advances in Intelligent Systems and Computing 612,
DOI 10.1007/978-3-319-61542-4_88

Most steps in this execution process have been discussed previously in many articles. Several contributions about task delegation [6], workflow decomposition [7] or service matchmaking [8] may be found. However, all these proposals are, at the end, based on a virtualized hardware platform which is implemented, traditionally, in a gateway or controller [5]. This component receives service invocations through a uniform interface and resolves the execution using the virtual hardware. This solution, nevertheless, does not meet the requirements of a real scenario, where real hardware devices with very heterogeneous and (most time) proprietary interfaces are included. Moreover, current systems tend to be ubiquitous, so hardware platforms present many redundancies (devices with the same functions). In this context, a solution to manage the service execution at low-level it is needed. The proposed solution should be able to collect information about hardware devices and, using it, decide the particular element which must execute each action (sending the corresponding order to it, using the adequate message and/or interface).

Therefore, in this paper a technology to manage the low-level service execution in CPS is proposed. The solution considers a first data collection process employed to determine the execution cost and the Quality-of-Service (QoS) offered by each device. Then, using a probabilistic selection procedure it is dynamically selected the device in charge of executing each action. Depending on the implementation, selected devices may delegate the execution (a paradigm usually named as choreography) or the hardware manager (see Fig. 2) must coordinate the entire process (orchestration).

The rest of the paper is organized as follows: Sect. 2 describes the technical proposal including a first general framework and a mathematical formalization. Section 3 includes the experimental validation of the proposed technology, and Sect. 4 shows the obtained results. Finally, Sect. 5 presents some conclusions.

## 2 Formalization of the Proposed Solution

In the most general scenario, Cyber-Physical Systems implement the architecture showed on Fig. 1(a). Each CPS acts as a networked autonomous system, connected with other CPS by means of the called Cyber-Physical Internet (CPI). Basically, the CPI includes a central service management infrastructure containing a service repository and a service manager. These components maintain the list of available services, publicly offered by CPS to be invoked from other CPS. The manager must (among other functionalities) determine (using, for example, semantic annotations [10]) if different services are really the same and response the queries about the stored services in the repository. The CPI, moreover, represents the backbone (usually the public Internet) which connects and communicates the different CPS among them and with the CPI infrastructure.

Then, each individual CPS relates with the CPI through an *Intersystem communication interface* which feeds the *Execution system*, together with the local *Domain expert environment*. Finally, a *physical platform* is in charge of the final execution. This layer may include different types of hardware devices (sensors, actuators, legacy systems, etc.) and different controllers. In this context, a service management solution for CPS should be composed of three different layers (see Fig. 1(b)). The *Connection level*

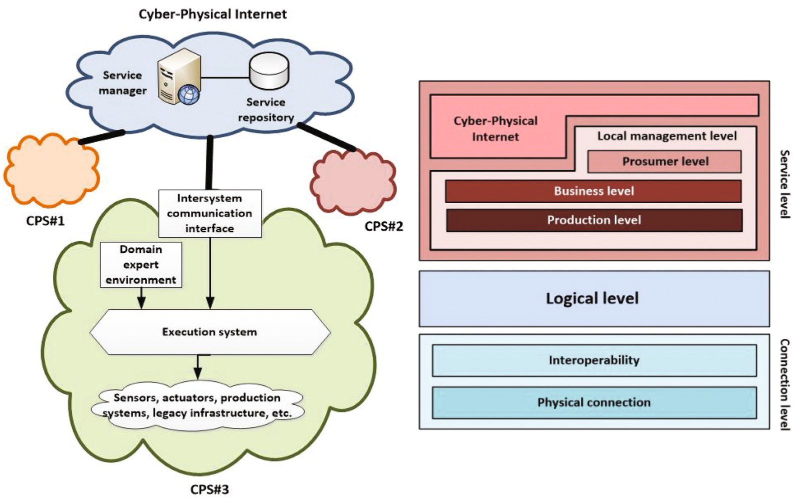

**Fig. 1.** (a) Typical CPS architecture (b) architecture of a service management solution in CPS

represents all the required procedures to integrate the peripherals with the central microcontroller of each device (*physical connection*) [9]; and the solution which connects each device with the other cyber-physical devices and with the hardware manager (*interoperability*). Interoperability mechanisms may be implemented in the microcontroller of each device or in an external broker (or hardware controller). The second level (*Logical level*) includes the coordination technologies employed by the hardware manager (see Fig. 2) to execute low-level services using the underlying hardware platform (or virtual components if also considered). Logical solutions include device selection, low-level service delegation, etc.

The proposed technology in this paper belongs to this level. Finally, *service level* integrates all the service-focused technologies which transform low-level services (sometimes called hardware services) into more abstracted entities nearest human comprehension (L*ocal management level*) and share resources publicly with other CPS through the CPI (*Cyber-Physical Internet*). Typically, three different service abstraction layers are considered: *production level* (which integrates services with a direct correspondence with hardware services), *business level* (representing services expressed into any technical executable language) and *prosumer level* (where services are described using any domain description language). As said above, mapping the previously described works (see the Introduction) into the proposed architecture for service management in CPS, it may be proved that the cited gap in CPS research corresponds exactly with the *Logical level*. Therefore, this paper describes a logical level technology. This technology includes three different procedures: hardware monitoring, service execution through individual management and service execution through group management.

## 2.1 Hardware Monitoring

The proposed technology follows the best effort paradigm, as it is considered that services do not include any indication about the QoS. However, in order to optimize the resource consumption, the proposed device selection for service execution it is not random, but it considers the current state of the hardware platform. Then a hardware monitoring process must be considered. Two types of information are included: the execution cost Q and a collection of hardware operation quality indicators Σ (such as availability). Figure 2 presents a functional architecture of the application scenario.

Information about hardware is provided by hardware controllers (the components which implement the *interoperability* functions), so costs are different for each service and location (i.e. hardware controller). Mathematically, then, costs are a set of sets (1)

$$Q = \{q^1, q^2, \ldots, q^N\} = \left\{\left\{q_1^1, q_2^1, \ldots, q_{M_1}^1\right\}, \ldots, \left\{q_1^N, q_2^N, \ldots, q_{M_N}^1\right\}\right\} \tag{1}$$

where $N$ is the number of hardware controllers and $M_i$ the number of services in the i-th hardware controller. Thus $q_j^i$ is the execution cost of the j-th service in the i-th location.

On the other hand, for each location and service, a set of values is employed as

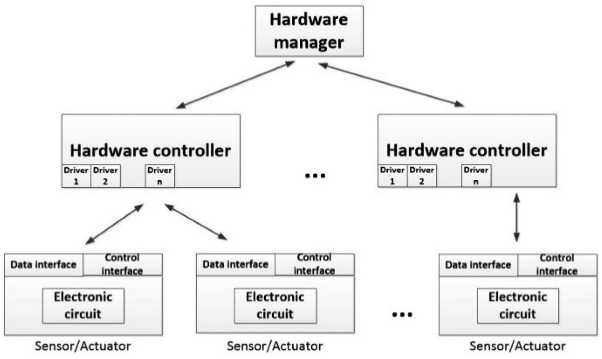

**Fig. 2.** Functional architecture of the application scenario

hardware operation quality indicators (2). Thus, mathematically, a triple nested list represents the quality indicators (3)

$$\sum\nolimits_j^i = \{\sigma_1^{i,j}, \sigma_2^{i,j}, \ldots, \sigma_R^{i,j}\} \tag{2}$$

$$\Sigma = \{\Sigma^1, \Sigma^2, \ldots, \Sigma^N\} = \left\{\left\{\Sigma_1^1, \Sigma_2^1, \ldots, \Sigma_{M_1}^1\right\}, \ldots, \left\{\Sigma_1^N, \Sigma_2^N, \ldots, \Sigma_{M_N}^1\right\}\right\} \tag{3}$$

where $R$ is the number of considered indicators. Quality indicators (such as response time or availability) may be directly measured, and/or present very well-known expressions [11]. However, the execution cost must be obtained through a proprietary procedure (4).

$$q_j^i = f_{cost}\left(q_{j_{user}}^i, q_{j_{hardware}}^i\right) \tag{4}$$

The cost of the j-th service in the i-th location is obtained through a cost function which considers two contributions: a cost specified by the administrator $q_{j_{user}}^i$ (employed to penalize certain equipment, do load balancing, etc.) and a hardware cost representing the resource consumption $q_{j_{hardware}}^i$. Depending if both contributions are independent or if they can compensate each other, the cost function may be a weighted arithmetic mean (5) or a geometric mean (6).

$$f_{cost}(\cdot,\cdot) = \lambda_1 q_{j_{user}}^i + \lambda_2 q_{j_{hardware}}^i \tag{5}$$

$$f_{cost}(\cdot,\cdot) = \sqrt{q_{j_{user}}^i \cdot q_{j_{hardware}}^i} \tag{6}$$

Finally, the hardware cost is obtained as a weighted mean of a collection of resource consumption indicators $\psi_k$ (such as the battery consumption or the required processing time). These indicators must be defined to take values in the interval $[0, 1]$, being 1 the value indication a higher cost (7).

$$q_{j_{hardware}}^i = \sum_k \beta_k \cdot \psi_k \tag{7}$$

In Sect. 3 we propose some particular examples of the named resource consumption indicators.

## 2.2 Service Execution Through Individual Management

Low-level services may be classified into various different classes attending to it geographical requirements (see Fig. 3).

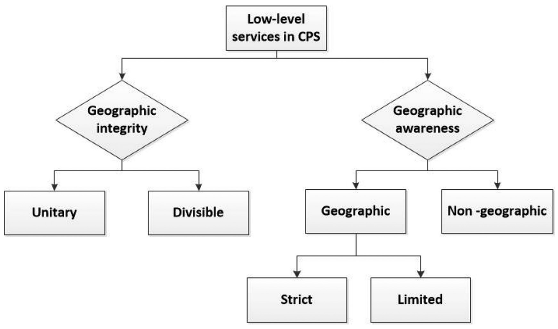

**Fig. 3.** Service classification depending on the geographic requirements

*Unitary services* are those which must be executed entirely in the same location (i.e. by the same hardware controller). Services which may be decomposed in parts to be executed in different controllers are *divisible*. On the other hand, *geographic services* must be executed in one particular location (*geographically strict services*) or in any of the location belonging to a certain group (*geographically limited services*). Geographic services must be complemented with metadata about the geographic restrictions. Non-geographic services may be executed in any location or hardware controller.

When a service invocation arrives to the hardware manager, it determines the group of locations $\mathcal{L}$ which may execute the service (depending on the geographic restrictions and the capabilities of each controller). Then, the set of locations $\mathcal{L}$ is considered as the sample space of a multidimensional random variable $\mathcal{X}$. Each one of the dimensions in this random variable corresponds with one quality indicator (see subsection A) or with the execution cost. Then the variable $\mathcal{X}$ presents $R+1$ dimensions. As physically all dimensions are independent, they are also statistically independent, and (then) the probability density function for the i-th service may be easily calculated (8).

$$f_{\mathcal{X}}(\ell_j) = \frac{q_i^j}{\sum_j q_i^j} \cdot \prod_{k=1}^{R} \frac{\sigma_k^{i,j}}{\sum_j \sigma_k^{i,j}} \tag{8}$$

In order to make all quantities comparable, all indicators and cost should be mapped into the same interval. Once built the probability function, different strategies may be followed to execute the service. In this section we are considering the hardware manager interacts individually with the hardware controllers.

Figure 4 represents various message sequence charts describing the whole procedure. We are explaining briefly each step:

1. The hardware manager selects the execution location (i.e. the hardware controller) following the probability function previously calculated.
2. The hardware manager sends the proper execution order to the selected hardware controller using the adequate data format and communication technology. Typically, a byte-oriented proprietary protocol [12] and low-energy communications (such as 802.15.4 [13]) will be employed.
3. If the selected hardware controller can execute the requested service, then the execution is successful in one step (Case A, see Fig. 4). It performs the proper actions and returns the result to the hardware manager.
4. If the execution order cannot be executed, but the controller has resources to manage the petition, two different possibilities appear. If, at the interoperability level, a choreography algorithm is implemented, then the hardware controller may delegate the service execution to other location (if exists). This secondary hardware controller (HC) will execute the service and will send the result to the hardware manager (case B). If choreography algorithms are not available, the selected hardware controller (hardware controller #k on Fig. 4) should send a rejection message to the hardware manager (HM). At this point, the HM acts as orchestrator and removes this HC from the list of possible execution locations $\mathcal{L}$, recalculates the probability function and performs a new selection (case C). This process may be repeated as many times as needed.

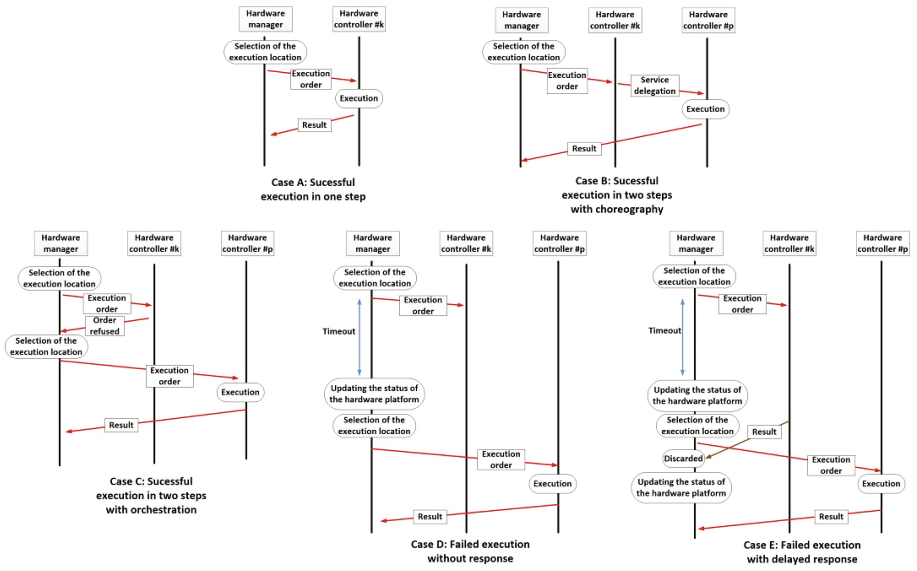
Hardware manager
Hardware controller #k
Hardware controller #p
Selection of the execution location
Execution order
Execution
Result
Service delegation
Order refused
Timeout
Updating the status of the hardware platform
Discarded
Case A: Sucessful execution in one step
Case B: Sucessful execution in two steps with choreography
Case C: Sucessful execution in two steps with orchestration
Case D: Failed execution without response
Case E: Failed execution with delayed response

**Fig. 4.** Message sequence charts (service execution through individual management)

5. Sometimes, HCs may get congested and execution orders cannot be neither performed nor managed. No response to the hardware manager is then sent. In order to resolve these situations, each time an execution order is sent, the HM activates a timer. If at the timeout no response is received (case D), the state of the hardware platform is updated (services offered by that HC are declared unavailable, the timeout is increased, etc.); and a new selection (removing the unavailable location) is performed.
6. Finally, in some occasions, the HC may send a result to the hardware manager after the timeout (case E). At the reception, this results is discarded (a new execution order was sent), but the hardware platform state in updated once more (declaring available the services of this HC but with a greater response time).

As main advantage of this solution, the execution effort is very balanced among all the available devices, and the execution time of a low-level service may be reduced to a minimum (for a successful execution in one step). However, this execution time is very variable, and grows much if it is necessary to order the same execution several times. Equation (9) represents a general expression for the mean execution time when using this first proposal.

$$\overline{T_{exe}} = (1-p)\left(2T_{com} + T_{proc}\right) + \sum_{k=1}^{n-1}(1-p)p^k\left(2T_{com} + T_{proc} + k\,\overline{T_{fail}}\right) + p^n n\,\overline{T_{fail}}\;\forall n \geq 2 \quad (9)$$

Where $p$ is the probability that a HC refuses or that it does not answer to an execution order. $T_{com}$ is the required time to communicate a HC and the HM. $T_{proc}$ is the needed time by a HC to perform the actions related to a certain service. And $\overline{T_{fail}}$ is

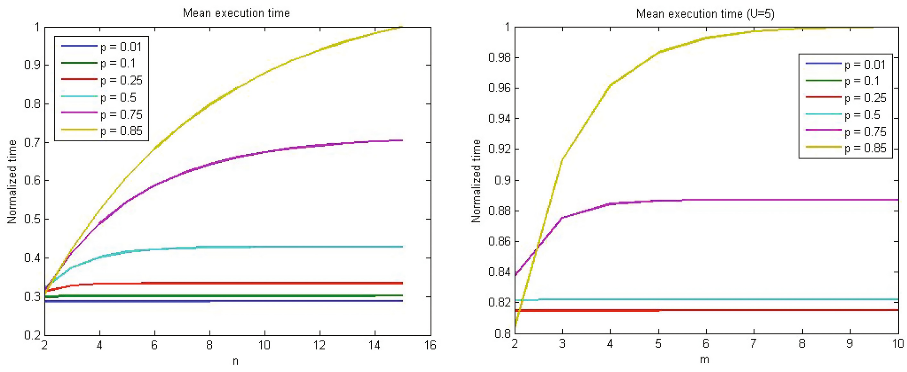

**Fig. 5.** (a) Graphical representation of the execution time: Eq. (9) (b) graphical representation of the execution time: Eqs. (10–12)

the mean time needed by the HM to notice a HC is not going to execute a service (for example, $T_{fail} = 2T_{com}$ if a rejection message is sent). Finally, $n$ is the number of HC which may execute the low-level service. Figure 5(a) represents graphically Eq. (9) for different values of $n$ and $p$. As can be seen, the execution time may grow up a 230% ($p = 0.85$). Some applications, however, do not tolerate these variations in the execution time, and a more stable solution is required. This problem is addressed with the second proposed algorithm.

## 2.3 Service Execution Through Group Management

The basic idea of this second procedure is to introduce a first additional phase, where the hardware manager asks various hardware controllers if they are able to execute a certain low-level service.

When a service invocation arrives to the HM, it selects a group of $U$ hardware controller which may execute this service (following the probability function described in the previous section). If the total amount of HC which may execute the service in the system is $V$, then a total of $\binom{U}{V}$ groups can be defined. Then (see Fig. 6) an execution petition is sent to all these HC. The hardware manager waits for the responses (a timer is launched in order to avoid blocking situations) which may accept the petition (offering some additional information, the expected processing time, for example) or reject the offer. Depending in the responses, the hardware platform state may be updated.

Finally, considering the HC which accept the execution petition it is constructed (in the HM) a new probability function in order to determine the device which must execute the service. From this point, the procedure continues as described in the case of a service execution through individual management

The additional initial phase makes the mean execution time greater, but much more stable (10–12). Figure 5(b) represents graphically the evolution of the mean execution time for different values of $m$ and $p$. As can be seen, in this case, there is only a 25% variation between the best and the worst situation (in the previous procedure the equivalent value was around a 250%), although times are higher.

$$\overline{T_{exe}} = (1-q)T_{success} + \sum_{k=1}^{m-1}(1-q)q^k\left(T_{success} + k\,\overline{T_{fail}}\right) + m\,\overline{T_{fail}}\,\forall m \geq 2 \tag{10}$$

$$T_{success} = (2T_{com} + T_{proc}) + 2T_{com} + T_{dec} \tag{11}$$

$$q = \prod_{k=1}^{U} p \tag{12}$$

Where $T_{dec}$ is the time employed by the HM to construct the additional probability functions, and $m$ is the number of groups which may be created with the $U$ HCs which can execute a service. Considering any of the proposed management procedures, if no hardware controller may execute the service there are two alternatives. If the service is unitary, then the invocation is refused and an error is returned to the higher level. However, if the service is divisible, then, the HM may decompose the service and employ any of the previously described procedures which its parts. Any case, if any of the parts cannot be executed, the entire invocation is refused and an error is returned by the HM.

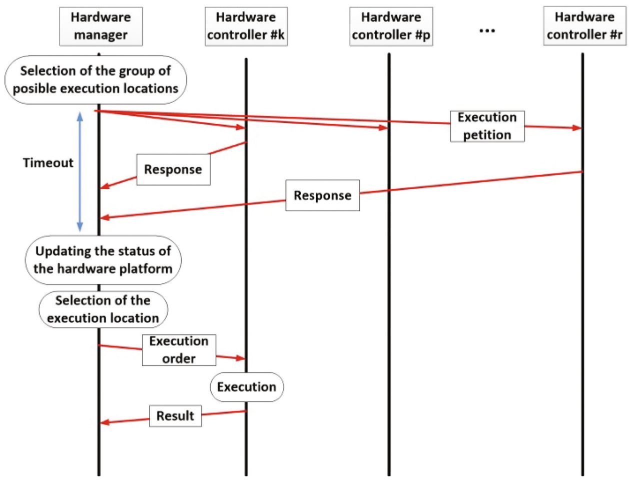

**Fig. 6.** Message sequence chart (service execution through group management)

## 3 Experimental Validation

An experimental validation was designed in order to validate the proposal and analyze the performance of the proposed technology. The experimental validation consisted of a simulated scenario based on the NS3 simulator. The deployed architecture contained one hardware manager, fifteen hardware controllers and one hundred and ten (110) sensors and actuators (homogeneously distributed among the hardware controllers).

In order to implement the described procedures in the components of the simulated scenario, each node in the NS3 simulator was provided with ghost node connecting with a tap bridge. These noes and bridges are able to connect an internal NS3 component with an external entity, which allows implementing in a very easy way complex algorithms and other proprietary solutions for which there are no libraries. In particular, the hardware manager and the hardware controller were connected with virtual Linux machines based on the KVM technology. Sensors and actuators were created by means of Simulink software and deployed in a new virtual Linux machine connected with the NS3 nodes.

Hardware controllers were connected with hardware manager by means of a Bluetooth backbone. An XML-based solution was employed as data format for this communications (HM-HC). The connection among sensors and hardware controller was resolved using the IEEE 1451 technology [14]. Figure 7 represents the planned scenario and the simulator configuration. All costs defined by users were fixed to the value of the unit $q_{user} = 1$. Contributions in the cost calculation were considered independent, so the cost function $f_{cost}(\cdot, \cdot)$ was defined as the arithmetic mean.

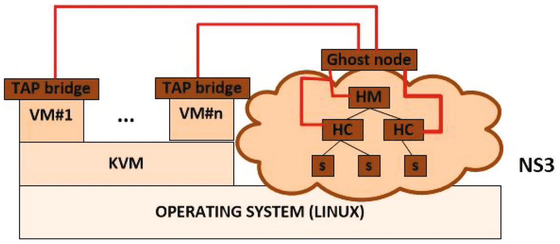

**Fig. 7.** Simulation scenario

The cost due to hardware considerations was understood as the arithmetic mean of two indicators: the battery consumption $\psi_{bat}$ and the occupied memory $\psi_{mem}$ (13, 14). All services in the system were defined as unitary and non-geographic.

$$\psi_{bat} = \frac{\textit{reduction in the battery charge}}{\textit{total battery capacity}} \tag{13}$$

$$\psi_{mem} = \frac{\textit{occupied memory}}{\textit{total installed memory}} \tag{14}$$

Finally, two different experiments were performed. During the first experiment connections among components were considered permanent, and losses in the Bluetooth backbone were increased progressively. The execution time was monitored when employing an individual management approach. During the second experiment, connections were considered ephemeral. The number of successful communications and the service execution time were monitored when employing both described procedures. Results were compared.

## 4 Results

In this section we present the results obtained in the experimental validation. Figure 8 (a) represents the results of the first experiment. As can be seen, the execution time grows monotonously. As the losses rate goes up, the number of necessary iterations to perform a successful execution grows, so the execution time increases. The variation is, as indicated in theory, near two magnitude orders. The evolution law is almost linear. On the other hand, in regular situations (networks without congestion) the execution time is very low (around 0.1 normalized temporal units).

In Fig. 8(b), the results of the second experiment compare the evolution in the execution time obtained for the first procedure (individual management) and for the

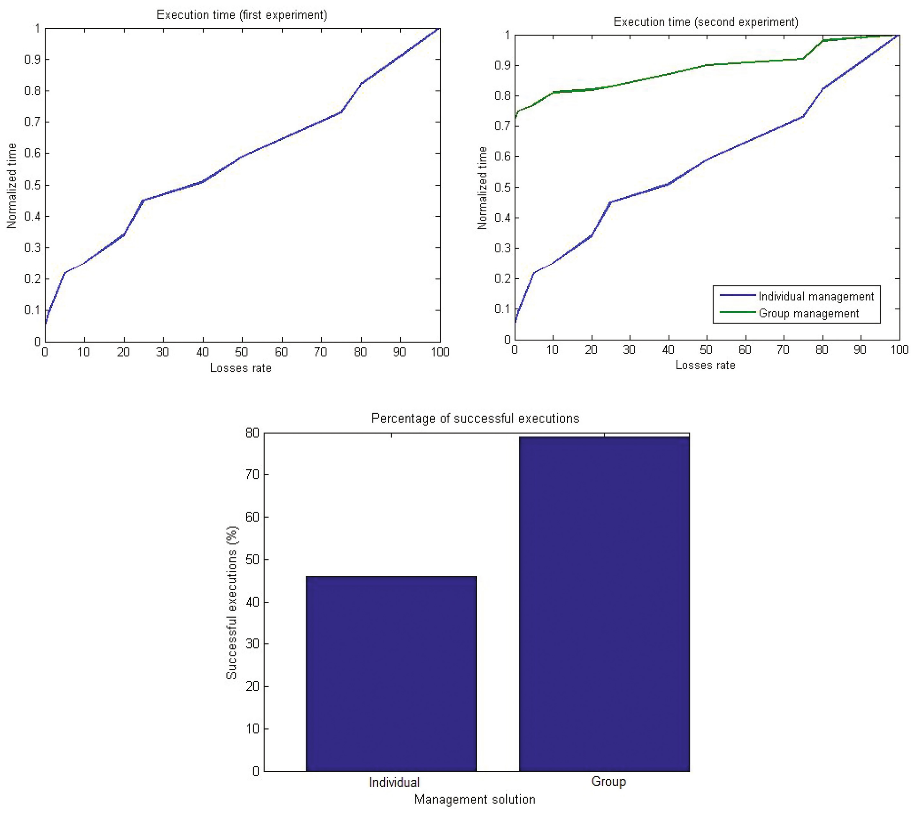

**Fig. 8.** (a) Results of the first experiment (b) results of the second experiment (execution time) (c) results of the second experiment (successful executions)

second one (group management). Groups of hardware controllers are made of five elements. As seen in theory, the execution time is higher when considering group management (in the best situation the execution time grows seven times). However, as showed on Fig. 8(b), the evolution law is almost constant, so almost no variations in the execution time are suffered.

This approach is very useful in scenarios involving ephemeral connections (as in mobile wireless sensor networks). These connections may tolerate a higher execution time, but variations are badly supported. Figure 8(c) shows the improvement in the number of successful communications (around a 50% increase).

## 5 Conclusions

Cyber-physical Systems require a solution for low-level service management. This solution must address the required procedures to finally associate a service execution with a hardware device.

In this paper we have proposed a low-level service management technology based on three different procedures. This solution belongs to the level named as “logic” in the proposed general framework. The first procedure considers a hardware monitoring process in order to evaluate the hardware platform state. The second one describes a service management and execution method based on individual management. Finally, a solution for service execution based on group managements is also proposed.

The experimental validation showed that individual management provides a very low execution time in the best case, but this time grows very fast if the situation of the hardware platform gets worse. In this cases a group management generates a higher execution times but much more stable.

**Acknowledgments.** Borja Bordel has received funding from the Ministry of Education through the FPU program (grant number FPU15/03977). Additionally, the research leading to these results has received funding from the Ministry of Economy and Competitiveness through SEMOLA project (TEC2015-68284-R) and from the Autonomous Region of Madrid through MOSI-AGIL-CM project (grant P2013/ICE-3019, co-funded by EU Structural Funds FSE and FEDER).

## References

1. Lee, E.A.: Cyber-physical systems—are computing foundations adequate? In: NSF Workshop on Cyber-Physical Systems: Research Motivation, Techniques and Roadmap, Austin, TX, USA, 16–17 October 2006
2. Rajkumar, R.R., Lee, I., Sha, L., Stankovic, J.: Cyber-physical systems: the next computing revolution. In: Proceedings of the 47th Design Automation Conference, pp. 731–736, June 2010
3. Sha, L., Gopalakrishnan, S., Liu, X., Wang, Q.: Cyber-physical systems: a new frontier. In: Machine Learning in Cyber Trust, pp. 3–13. Springer, New York (2009)

4. National Institute of Standards and Technology: Cyber-physical systems homepage. http://www.nist.gov/cps/. Accessed 17 Mar 2017
5. Bordel Sánchez, B., Alcarria, R., Sánchez-de-Rivera, D., Sánchez-Picot, A.: Enhancing process control in industry 4.0 scenarios using cyber-physical systems. J. Wireless Mob. Netw. Ubiquitous Comput. Dependable Appl. **7**(4), 41–64 (2016)
6. Horváth, K.A., Telek, M.: Task delegation in a peer-to-peer volunteer computing platform. In: International Conference on Analytical and Stochastic Modeling Techniques and Applications, pp. 115–129. Springer International Publishing, May 2015
7. Jaradat, W., Dearle, A., Barker, A.: Workflow partitioning and deployment on the cloud using orchestra. In: Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, pp. 251–260. IEEE Computer Society, December 2014
8. Kritikos, K., Plexousakis, D.: Novel optimal and scalable nonfunctional service matchmaking techniques. IEEE Trans. Serv. Comput. **7**(4), 614–627 (2014)
9. Bordel, B., de Rivera, D.S., Alcarria, R.: Plug-and-play transducers in cyber-physical systems for device-driven applications. In: 2016 10th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), pp. 316–321. IEEE, July 2016
10. Talantikite, H.N., Aissani, D., Boudjlida, N.: Semantic annotations for web services discovery and composition. Comput. Stan. Interfaces **31**(6), 1108–1117 (2009)
11. Shenker, S.: Specification of guaranteed quality of service (1997)
12. Bordel Sánchez, B., Alcarria, R., Martín, D., Robles, T.: TF4SM: a framework for developing traceability solutions in small manufacturing companies. Sensors **15**(11), 29478–29510 (2015)
13. Callaway, E., Gorday, P., Hester, L., Gutierrez, J.A., Naeve, M., Heile, B., Bahl, V.: Home networking with IEEE 802.15.4: a developing standard for low-rate wireless personal area networks. IEEE Commun. Mag. **40**(8), 70–77 (2002)
14. Lee, K.: IEEE 1451: a standard in support of smart transducer networking. In: Proceedings of the 17th on Instrumentation and Measurement Technology Conference, IMTC 2000, vol. 2, pp. 525–528. IEEE (2000)