Managing Dynamicity in SoS

Sara Bouchenak[1](✉), Francesco Brancati[2],
Andrea Ceccarelli[2], Sorin Iacob[3], Nicolas Marchand[1],
Bogdan Robu[1], and Patrick De Oude[3]

[1] Université Grenoble Alpes, Grenoble, France
sara.bouchenak@insa-lyon.fr,
nicolas.marchand@gipsa-lab.fr,
bogdan.robu@gipsa-lab.grenoble-inp.fr
[2] Department of Mathematics and Informatics,
University of Florence, Florence, Italy
francesco.brancati@resiltech.com,
andrea.ceccarelli@inifi.it
[3] Thales Netherlands B.V., Hengelo, The Netherlands
{sorin.iacob, patrick.deoude}@nl.thalesgroup.com

1 Introduction

SoS dynamicity refers to short-term changes in an SoS, which occur in response to changing environmental or operational parameters of the CSs. These changes may have different effects, such as SoS adaptation or the generation of emergent phenomena. This chapter starts by recalling the MAPE approach in Sect. 2 before to introduce existing monitoring approaches in Sect. 3. Finally, Sect. 4 overviews existing reconfiguration techniques for SoS dynamicity management, related to Analyzis, Planning and Execution phases and illustrates through an examples the possible implementations of dynamicity management with modelling and feedback control techniques.

2 Overall MAPE Approach

We follow the classical MAPE-K control loop for designing an autonomic manager over managed elements (Fig. 1). This consists mainly in components to Monitor, Analyze, Plan and Execute the reconfiguration plan. When an SLA (Service Level Agreement) and its associated service level objectives are associated with the service of a managed element, the MAPE control loop guarantees that those service level objectives are met, and if it is not the case, a new plan is calculated and used to reconfigure the system.

This work has been partially supported by the FP7-610535-AMADEOS project.

A. Bondavalli et al. (Eds.): Cyber-Physical Systems of Systems, LNCS 10099, pp. 186–206, 2016.
DOI: 10.1007/978-3-319-47590-5_7

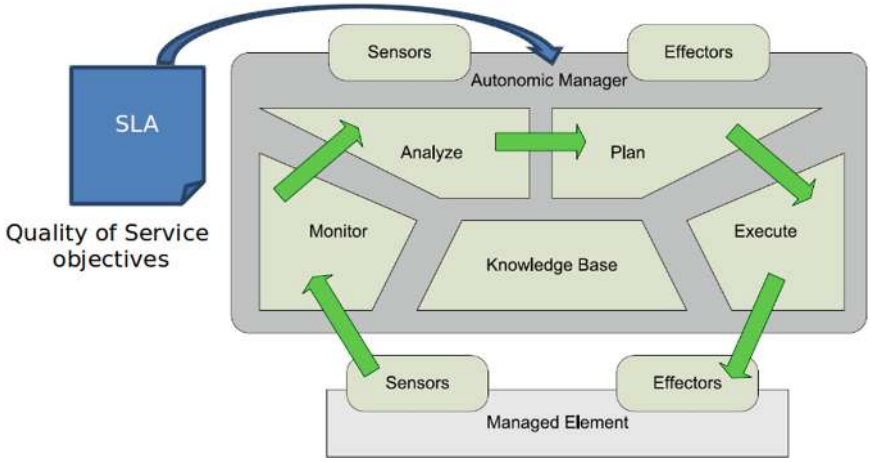


Fig. 1. MAPE control loop

2.1 SoS Management Infrastructure

In this section we present how to exploit the AMADEOS SysML profile described in Chap. 4 in order to build the infrastructure for MAPE purposes. To this end, we implemented through the profile six MAPE architectural patterns. Thus, for each of the six patterns, we realize a SysML block diagram built using the stereotypes defined in the profile.

Each CS is represented as a block and it has an interface, either a RUMI or a RUPI interface in order to enable the exchange of messages and physical entities respectively. Each CS may implement the SoS management activities, namely monitoring, analysis, planning and execution.

In the following we report the (1) Hierarchical Control, (2) Master/Slave, (3) Regional Planner, (4) Coordinated control, (5) Information sharing, (6) Atomic patterns and a conclusive analysis on their recursive inclusion in a SoS.

2.2 Hierarchical Control Pattern

In the hierarchical control pattern, we have a managed CS which is controlled directly by a managing CS by means of their RUMI interfaces over which monitoring information and the enacting actions are transmitted. As we can notice (Fig. 2), the managing CS has been stereotyped with all the functions of the MAPE cycle. We present below the UML representation of such pattern.

2.3 Master/Slave Pattern

In the master/slave pattern¸ a set of managing CSs shares part of the MAPE functions (Fig. 3). In our instantiation, master CS (Managing_AP_CS) performs analysis and planning activities and two slave CSs perform monitoring and execute functions

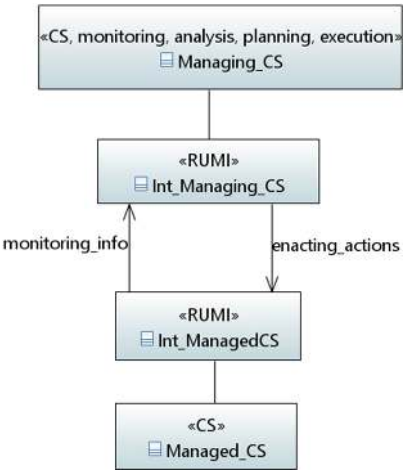


Fig. 2. Hierarchical control pattern

(Managing_ME_CSx). The slave CSs send monitored_info to the master and they receive planned info from the master CS. This information is exchanged through the RUMI interfaces of slave CSs and the master CS. Finally, the slave CSs are in charge of communicating each with a managed CS. The latter transmit monitored data to the slave CS, which in turn forwards back the enacting actions as planned by the master CS.

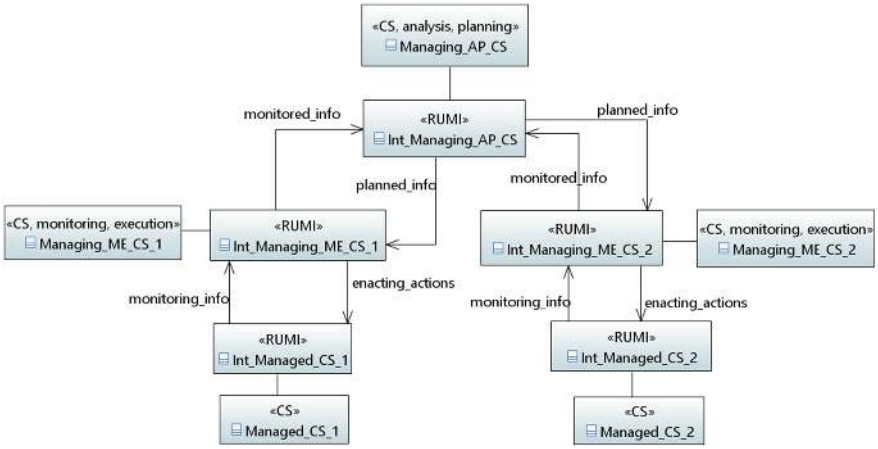


Fig. 3. Master-slave pattern

2.4 Regional Planner Pattern

In the Regional Planner, a regional managing CS (Managing_P_CS_x) implements only the Planning activity (Fig. 4). Instead, Analysis, Monitoring and Execution are

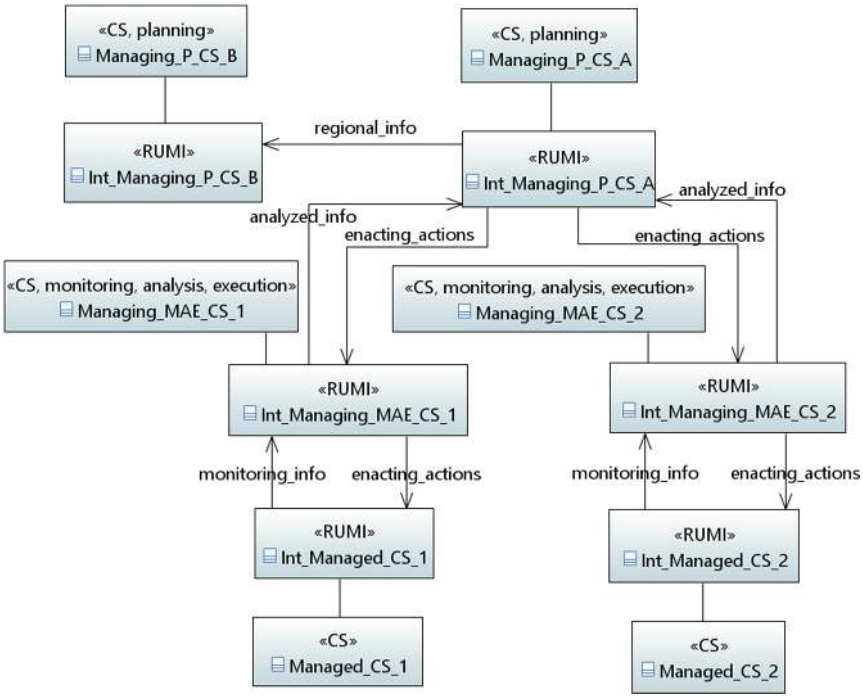


Fig. 4. Regional planner

delegated to other managing CSs (Managing_MAE_CS_x). The regional managing CS is responsible for a region of CSs and it exchanges with other peer CSs the information on a regional basis. At the bottom level, each managed CSs (Managed_CS_x) send monitoring information to its corresponding managing CS (Managing_MAE_CS_x), which performs analysis activities and then forwards the results to the regional CS (Managing_P_CS). The latter performs the Planning and forward enacting actions back to the managing CS. Finally, the managing CS enacts such actions towards the managed CSs.

2.5 Coordinated Control Planner

In the following, we present the implementation for the non-formal hierarchy pattern Coordinated Control pattern (Fig. 5). The coordinated control pattern consists of a set of CS implementing all the MAPE phases (Managing_CS_x) and exchanging through their RUMIs the monitored, analysis, planned and execution information. Through RUMIs, the managing CSs can collect monitoring info from the managed CS (Managed_CS_x) and forward the actions to enact.

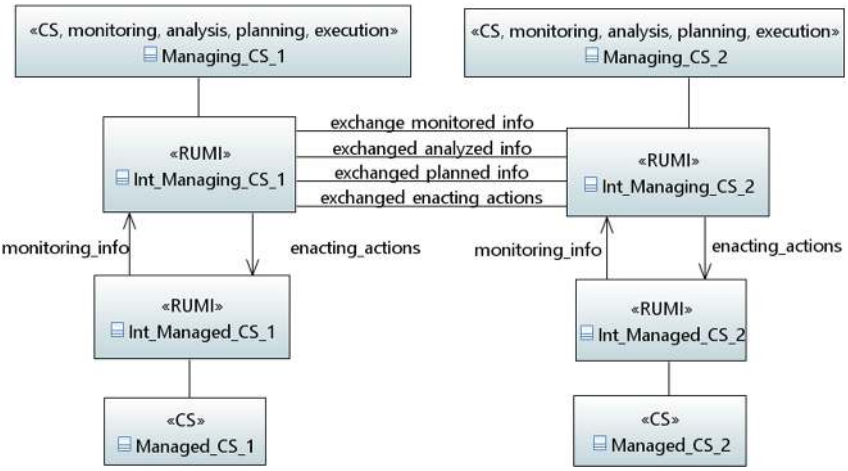


Fig. 5. Coordinated control

2.6 Information Sharing Pattern

The information sharing pattern is another non-formal hierarchy pattern similar to the coordinated control pattern (Fig. 6). The only thing that differentiates the corresponding implementations is the nature of information which is exchanged through the RUMIs of the managing CSs. In the sharing information pattern only monitored data are exchanged while the rest of information is not shared among managing CSs (Managing_CS_x).

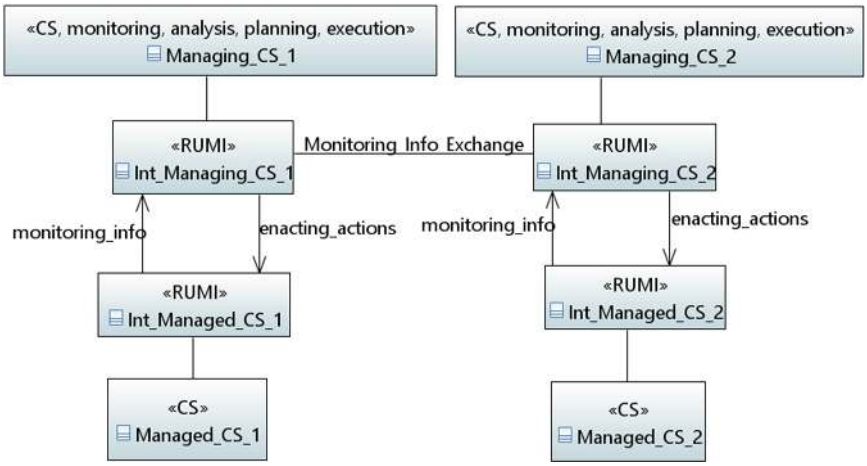


Fig. 6. Information sharing

2.7 Atomic Pattern

In the following, we present the atomic pattern to enable the interaction of a CS with the physical environment (Fig. 7). To this end, the managing CS (Atomic_CS) carries out the MAPE cycle once it has received monitoring information through its RUPI interface and it forwards physical signal over the same interface. Physical entities received through the RUPI interface come from the afferent environment while the outgoing flow of physical entities is forwarded to the efferent environment.

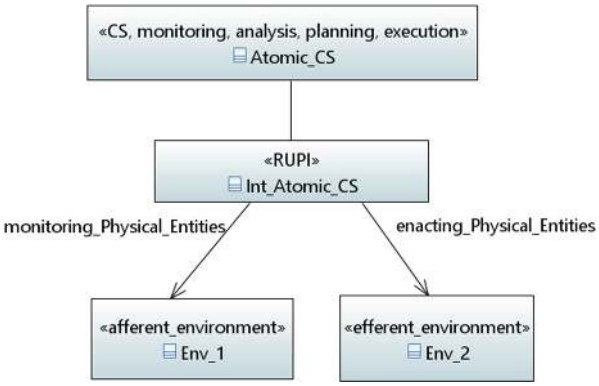


Fig. 7. Atomic pattern

2.8 Recursive Inclusion of Patterns for SoS Design

At SoS level, the MAPE should be applied recursively. Each CS implementing part of the MAPE may in turn be substituted recursively by another pattern which is realized through a further set of CSs. This approach foresees the possibility to have several nested levels of MAPE, each of them belonging to a different hierarchical level of the SoS, or holarchical set of CSs. Distinct set of CSs are observed by a MAPE in which the Adaptive Monitoring, the Analyzer and Planner, and the Reaction Strategies coexist and cooperate.

Let us consider the scenario depicted in Fig. 8, in which several CS are represented and controlled by an implementation of the MAPE-K cycle in which we have a dedicated CS for Monitoring, another for Analysis and Planning and a third one for Execution.

The Adaptive Monitoring (M) is able to detect events according to the QoS specifications of the CS. The Cognitive and Predictive Models (AP) search for the causes and the effects correlating data incoming from the Adaptive Monitoring. The Reaction Strategies (E) perform the proper recovery action. The considered SoS can be integrated in a more complex SoS; Fig. 8 gives a representation of the recursive foreseen architecture.

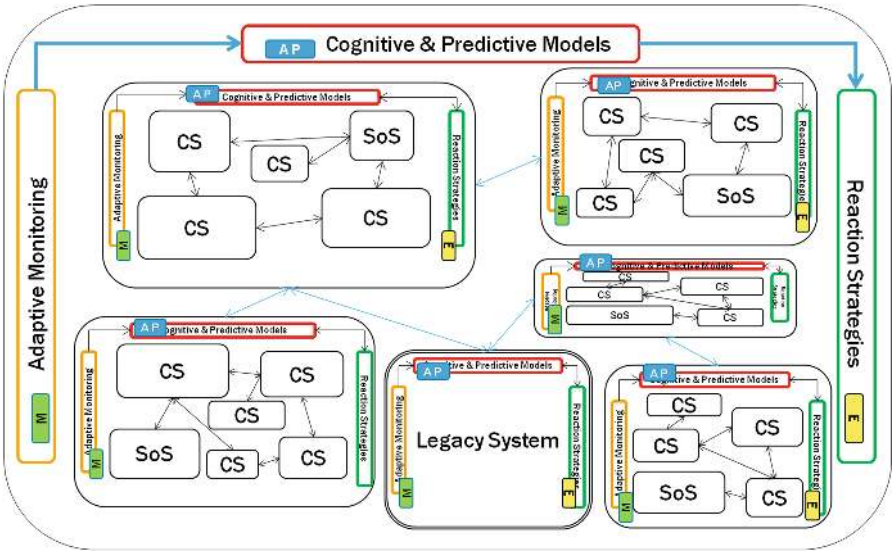


Fig. 8. Recursive view of the AMADEOS architecture

3 Monitoring/Analysis

In this section, we report on the broad topic of monitoring in SoSs. In particular, focusing on observation and data analysis (i.e., the Monitoring and Analysis components of the MAPE building block). We investigate basics on monitoring and detection (Sect. 3.1) and main monitoring approaches (Sect. 3.2).

3.1 Basics on Monitoring and Detection in SoSs

It is very important to guarantee that an SoS behaves as expected. To this end, monitoring activities control the SoS by means of verifying that system behavior and performance comply with well-defined rules. Verification activities can be carried out at two different stages either on-line, i.e., while monitoring data are collected, or off-line i.e., after the collection process.

In monitoring literature, the system which has to be monitored is called target system while the hardware component and the software application within the target systems are called respectively target component and target application. In our case a CS represents the target system while the target component and the target application are represented by the physical and software part of the CS itself. We refer to the CS which receives monitored information as monitoring CS.

Monitoring activities consists in observing behavior and performances of CS target components in order to collect useful information to guarantee the correct SoS functioning. Monitoring activities by themselves are not sufficient to guarantee the correct behavior of an SoS, but they have to be integrated with techniques to diagnose the SoS

behavior in its execution environment. To this end, we have presented in deliverable D3.1 [1] an SoS Management Infrastructure which complements Monitoring with Analysis, Planning and Execution (MAPE) facilities. This section focuses on the Monitoring (M), and partially also on the Detection of events (Analysis of data), consequently addressing the MA letters of the MAPE loop. These two actions (MA) are in fact often tightly bounded, because a monitoring system is usually conceived and instantiated with the specific intention of detecting specific events or verifying that certain conditions are met.

Let us now focus on the way monitoring activities are carried out. Essential objects that are exploited to monitor the system behavior are the so called probes. Probes can be inserted either inside or outside the target system and provide useful information on how the system behaves. As an example Fig. 9 shows the probes inserted within the target system which can also provide the system intermediate output (see Fig. 9-b) and it shows the possibility of monitoring the system as a black box (see Fig. 9-a).

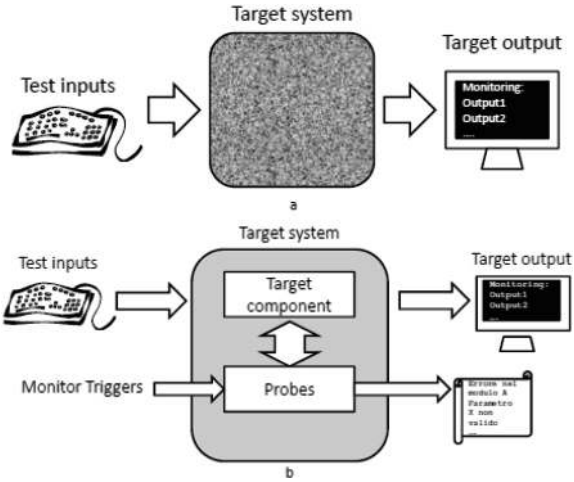


Fig. 9. Black box (a) and instrumented (b) monitoring of the target system

Probes can be hardware or software. In the first case hardware signals are monitored, while in the second case, code is inserted within the target application to collect internal information of the system (code instrumentation). Two rules have been defined which have to be respected by probes:

- they should observe as much information as necessary to satisfy the objectives of the monitoring activities,
- they should not compromise, or at least compromise as little as possible, the behavior of the target system.

3.2 A General Approach to SoS Monitoring

Considering the complexity and heterogeneity of an SoS, it is necessary that a monitoring solution deals with the issue of where to deploy the monitoring and detection system. It could be placed locally on each CS or globally at a higher level. Both approaches have advantages and disadvantages with respect to the CS and/or the SoS, as listed in the following:

- local solution pros: allows to perform a more precise detection activity on the single CS due to the perfect knowledge of CS itself.
- local solution cons: could negatively affect the performance of each CS, thus compromising the overall performance of the SoS.
- global solution pros: allows to improve detection accuracy, like detection of detrimental emergence phenomena.
- global solution cons: requires a large amount of data to be transferred from each CS, thus potentially affecting the network bandwidth in negative way.

To capture the pros of both the local and global solution, we envision the following overall architecture.

Each CS describes the provided services to the other CSs through the RUI specification. The interface models are part of the RUI specification and must be based on an agreed ontology explaining the meaning of the interface variables exchanged across the RUI and must be compatible with each other. In order to establish the desired quality of service (QoS), quality metrics must be expressed as well in the RUI specification. For example, a Service Level Agreement (SLA) should be negotiated between the service provider and requester. Thanks to the RUI specification, the monitoring and the detection systems can ignore the intrinsic characteristics of the CSs, but they are aware of its quality metrics and its SLA.

Each CS that is included in the SoS can be equipped with a Local Detection System (LDS). The LDS (i) includes probes exposed through the RUI and that are necessary to observe events; (ii) if necessary, implements the atomic pattern to manage the physical environment of the CS itself. The knowledge of the LDS is limited to the CS: in other words, the other connected CSs are ignored. The different detectors, relying on the exposed probes, can be organized and coordinate following the different MAPE patterns.

At SoS level, a global detection system will be also deployed. It differs from the ones deployed at a local level because it has an overall view of the SoS and consequently it has the ability to observe and detect events as a combination of the outputs of the individual LDSs. Furthermore, it may consider different and additional quality metrics and indicators with respect to the LDSs. The global detection system fetches data from the LDS or MAPE instantiations available in the SoS, and perform global monitoring and analysis, acting according to the master/slave pattern.

Figure 10 shows a high level representation of the architecture of the envisioned system. Specifically, the CSs composing the SoS are represented in different shapes, to show that they are different one from the other. Each CS is able to communicate with the others by means of the RUI interface, represented by the box labelled RUI. Finally, the local detection systems, labelled LDS are also included in the representation of each CS, in order to eventually detect anomalous events on the corresponding CS. At the top

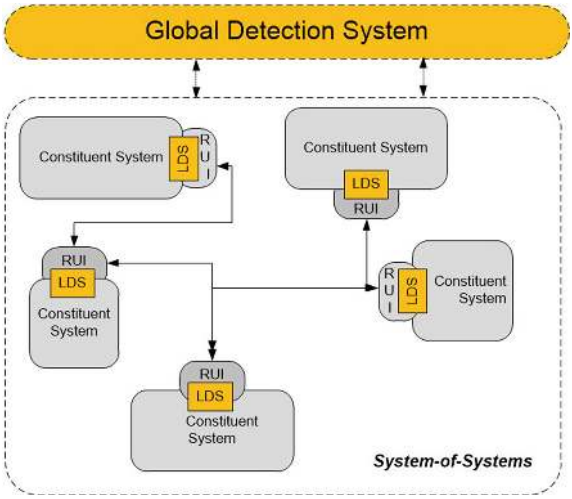


Fig. 10. Monitoring infrastructure in SoSs

of the figure, the Global Detection System is also shown, which observes the status and the events that are happening at the SoS level.

Depending on the monitoring purposes, confidentiality and privacy issues may need to be guaranteed through proper data security and anonymization. Especially, while anonymization solutions can be executed locally, confidentiality requires that the endpoint agrees on adequate secure communication protocols.

4 Analysis/Planning/Execution

4.1 Overview

The main challenges related to the dynamic adaptation of CPSoS stem from the distributed nature of the measurement and control infrastructure (MAPE).

Since the Monitoring and Analysis blocks have been discussed earlier, this section focuses primarily on a potential design of the Planning and Execution blocks.

The M and A functions of the MAPE building block determine the values of the CPSoS parameters, whereas the P and E functions close the control loop either by generating control signals for the CS, or by adjusting the environmental parameters of the CS, as depicted in Fig. 11.

All MAPE functions are instantiated for a particular CPSoS dynamicity model, which specifies which state parameters need to be measured, what metrics have to be used for combining or aggregating these parameters, how the control must be implemented to achieve the desired effect, and how this control algorithm generates CS control parameters. Obviously, this domain knowledge applies in a similar way for all the composite MAPE patterns described in the AMADEOS deliverable D3.1 [1].

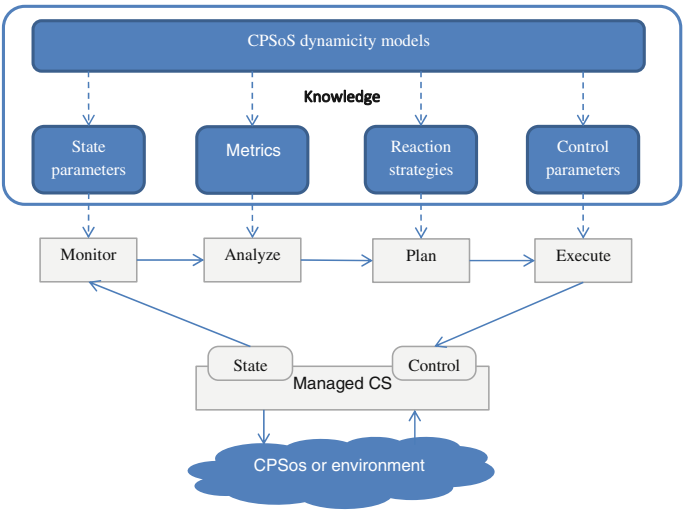


Fig. 11. Close loop control involving a CS and a MAPE block

To emphasize the control aspect, we redraw the MAPE control loop as in Fig. 12. Although not fundamentally different from a traditional control loop, Fig. 13 emphasizes the potential difference between the input and output parameters ($Y(t)$ and $X(t)$, respectively) toward the CPSoS and the monitoring and control parameters ($\mu(t)$, and $\varepsilon(t)$, respectively) toward the MAPE blocks. This is useful for CPS, since the monitoring and control of the RUPI-based interactions can also be executed through RUMI, which adds flexibility to the definition of reaction strategies. The ($\oplus$) blocks denote some suitable transformations that map the external and control input vectors onto a common metric space, and the $\oplus$ block combines these values in a single metric to obtain the control value. Based on this value the Control block generates a vector $\sigma(t)$

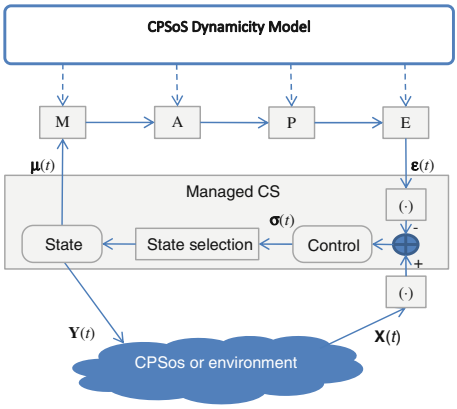


Fig. 12. Control and feedback parameters in MAPE-based control loops

whose elements show the mismatch between the current and desirable state parameters, so a new state can be selected that compensates for this mismatch.

4.2 Example: Control Loop for Electrical Vehicle System

If the CS is a Charging Point (see AMADEOS use case [2, 3]) Y(t) in Fig. 13 can be a state vector specifying the charging current, maximum allowed power phasor variations, and last kWh price. When an Electric Vehicle (EV) is connected to the CP, the State includes, among other functions, a metering function which could generate the monitoring vector μ(t) as a sequence of messages at regular time intervals, containing instantaneous measurements of the current drawn by the EV, and the maximum available current for that CP. Similarly, X(t) could be the remaining charging time as estimated by the EV, and ε(t) could be a sequence of asynchronous (i.e. event-triggered) messages containing a new value for the kWh price, and a new maximum power rating for the CP.

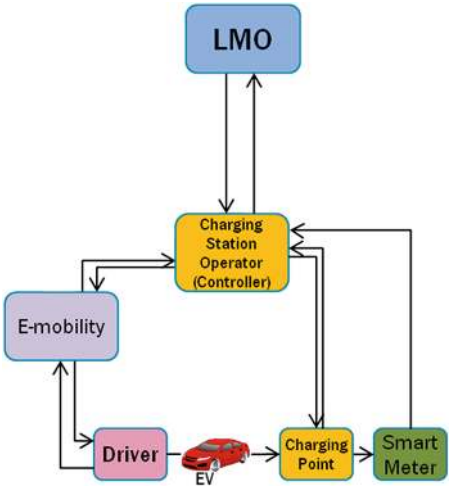


Fig. 13. Simplified functional diagram for an electrical vehicle charging station

Obviously, the summation block in Fig. 12 assumes an appropriate abstraction of the two inputs (MAPE controls and external CPSoS inputs), which is achieved through some mappings. In the example considered above both the remaining charging time and the new kWh price could be mapped on some real value expressing the economic efficiency of the CP.

To illustrate several reaction strategies, we construct a simple example derived from the AMADEOS EV Charging use case. Assume that a Charging Station includes ***N*** charging Points (see Fig. 13), has a total charging capacity Q, and a maximum charging rate I¬max. Each of the CP has a maximum charging rate ICP_{maxICP}, with $NI\_CP_{max} = aI_{max}$, and a > 1, which means that when all CPs are in use, not all of

them can deliver the maximum charging rate. To compensate for this limitation, the CP can influence the charging demand by increasing the energy unit prices. In general, the energy unit price has to be agreed between the EV user and the CSO before the charging operation starts, and should not be changed until the charging is complete. For this reason, it is convenient to define a charging contract in terms of total requested electric charge by user i, Q_i, and the maximum charging duration t_i. For simplicity, we assume that the minimum charging times only depends on the maximum current that a CP outlet can provide.

If the CSO chooses to guarantee a constant charging rate I_i, then the charging time for EV i will be constant as well (within some uncertainty limits): $t_i = \frac{Q_i}{I_i}$. Although it is both in the interest of the CSO and the user to minimize t_i, this is not achievable simultaneously for all the CPs.

A good pricing strategy should try to approach the maximum charging current of the CSO(I_{max}), without lowering the charge unit price below a given minimum. As an example one could consider the following pricing function by the CSO, for user i:

$$p_{CSOi}(Q_i, t_i) = p_0 \frac{t_{max} - t_{imin}}{t_i - t_{imin}} \left(1 + \frac{1}{I_{max} - I}\right), t_{min} < t_i \leq t_{max}, I < I_{max} \quad (1)$$

The minimum charging time t_{ijmin} is achieved for the maximum available charging rate given the already committed capacity (I) for the CSO : $t_{imin} = \frac{Q_i}{I_{max} - I}$. The third term of the product in the above formula increases the price as the committed capacity approached the maximum.

Overall, the price variation looks like the graph in Fig. 14.

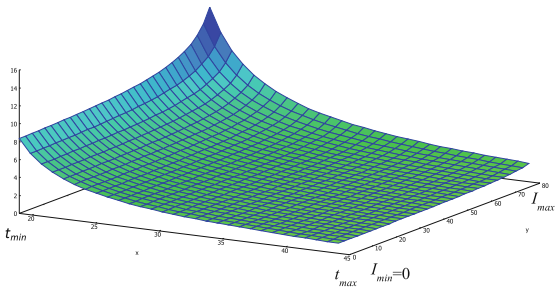


Fig. 14. Possible price adaptation as a function of requested charging time and available

At the same time, a user will always choose a shorter charging time, provided that the price does not increase beyond a predefined personal limit p_{imax}:

$$p_i = \min(p_{imax}, p_{CSOi})$$

Of course, this is a naïve view, since I_j varies in time as other running charging actions end (see Fig. 15), so a significant part of the charging capacity is not used. The

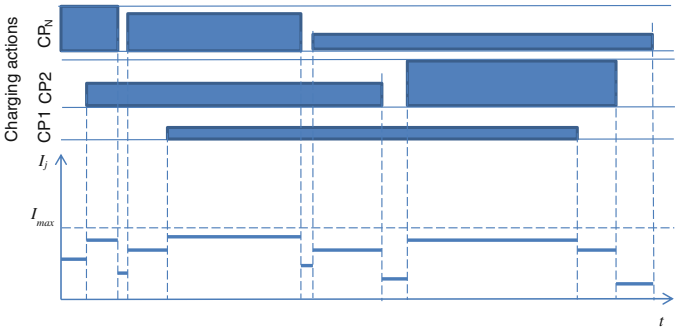


Fig. 15. Variation of the total charging current at CSO

total revenue for the CSO at any given time $P(t) = \sum_{i=1}^{N} p_i(t)$. At the same time, the instantaneous cost for the CSO is given by a component proportional to the total current absorbed plus some constant cost value c_0: $C(t) = c_0 + cI(t)$. The profit made by the CSO can thus be expressed as:

$$\rho_{CSO}(t) = P(t) - C(t) \tag{2}$$

By allowing a variable maximum charging current, the actual charging time results shorter than the one calculated with the pricing model for constant charging current used in this example. The statistics of charging times and new requests will require pricing strategies that take into account the expected variations in the occupancy of CPs.

4.3 Analytic Approaches

When the CPSoS behaves according to a known model expressed analytically, the control can be defined in terms of this model. The pricing model in Eq. (1) allows the CSO to set a dynamic price for a charging operation at constant current. However, this model leads to unused capacity, so it may be advantageous for the CSO to deliver a faster charge if there is unused capacity and other requests are expected to arrive soon.

An optimal control problem with infinite time horizon attempts to maximize some cost or benefit function, such as the one in Eq. (2) [4]. A potential objective function for the optimal control problem could attempt the simultaneous optimisation of the following aspects:

- minimizing the agreed charging time for the already started charging operations.
- maximizing the price for the new charging contract.

The control parameters are the individual charge unit prices and the charging currents for each user. The constraints are the total current for the CSO, the maximum

current of each CP, and the committed charging time for the already started charging actions.

The first term of the objective function defines how the charging current for user I can be increased after the charging action for user j ends. This can be done for instance, proportionally:

$$\Delta I_i^+ = I_j \frac{I_i}{I} = I_j \frac{I_i}{\sum_{k \neq j} I_k} \tag{3}$$

Whenever a new request comes at a time t_{i_start} after the adjustment of the charging rates, the new charging current must be maximised by reducing the current for the still running charge operations, down to the limit that would still allow the completion of the charging within the remaining time according to the original contract for user i:

$$\Delta I_i^- = \frac{Q_i - \int_{t_{i\_start}}^{t_{j\_start}} I_i(t)dt}{t_{i\_end} - t_{j\_start}} \tag{4}$$

where t_{i_start} and t_{i_end} are the starting and ending times of the charging operation for user i.

Equations (1) to (4) can be used for defining different control approaches, such as Optimal (Stochastic) Control with infinite time horizon [4], Linear Quadratic Controls [5], etc.

4.4 Machine Learning Approaches

In case when the dynamic behaviour of the CPSoS is unknown, or cannot be expressed analytically, data-driven techniques can be used for obtaining an implicit "encoding" of system behaviour. Such an encoding should be able to predict the outputs the system will generate for a given input and contextual parameters.

Machine learning includes a set of techniques that use observations of a system's behaviour patterns to attempt predicting its future states. If we consider the predictive or supervised machine learning approach the goal is to learn a mapping from input values $\mathbf{X}$ to an output value y based on a set of input-output pairs called a training set. The mapping can be represented as a function f_Θ, with a set of parameters Θ, that can be used, given an unseen before pattern $\mathbf{X}_i$, to predict $\hat{y}_i$, i.e. $\hat{y}_i = f_\Theta(\mathbf{X}_i)$. The function f_Θ is what we call a model that is parametrized with a set of parameters Θ. f_Θ can be, in fact, a simple linear function (regression), but generally is a complex algorithm that maps inputs to output values. The goal of machine learning is to find the set of parameters Θ of a chosen model based on a training data set D using a learning algorithm. If the parameters Θ are properly learned then we will be able to estimate the class label $\hat{y}_i$ based on certain unseen input values xj $\notin$ D, where $\hat{y}_j = y_j$ in the majority of the cases.

Artificial Neural Networks (ANN) implement a machine learning technique inspired by a simplified model of the working of the brain. The elementary operations in an ANN are weighted summations of the inputs to obtain an output value. Different

output values are obtained by summing the same input with different weights. This way, an input vector is mapped to an output vector. A multi-layer NN combines usually two or more such mapping units. The training of an ANN attempts to adjust the weights such that a particular output is consistently obtained for different input patterns belonging to the same class. When this is achieved, the ANN is said to be able to generalize. Another aspect of training attempts to adjust the weights such that different output patterns are generated for input vectors belonging to different classes. When this is achieved, the ANN is said to be able to discriminate. Training algorithms have been developed that achieve a good trade-off between these two properties. One such algorithm is called the back-propagation learning algorithm that uses an iterative scheme, such as gradient descent [6], to optimize Θ in the learning equation described earlier.

Returning to the EV charging example we want to estimate the best price for a new user i by learning a model based on the following input variables:

- the number of charging points N;
- the requested electrical charge Q_i by user i;
- the maximum charging duration t_{imax} requested by user i;
- the maximum charging durations of the other users $\mathrm{j}(\mathrm{j} \neq \mathrm{i})$;
- the current charging rates I_j of other EVs;
- the maximum charging rate for the CPs;
- the maximum charging rate of the CSO;
- the minimum charging time of EV i, t_{imin};
- the expected number of users in the coming hour (based on historic data);

Let's denote these input variables with x_n. Based on x_n we want to learn a model from which the best price $\hat{p}_i$ can be estimated for a new user i. In order to learn such a model the following cost function will be used

$$J(\theta) = \frac{1}{2}\sum_{n=1}^{N}(p_{ni} - \hat{p}_{ni})^2 \tag{5}$$

Where $\hat{p}_{ni}$ is the estimated price by the model, p_{ni} is the price associated to the input variables, N is the total number of training samples and n corresponds to the nth training sample $(x_n, p_{ni}) \in D$. In case an ANN is used as a model to learn the best price the gradient descent back-propagation algorithm can be used as an iterative scheme to optimize the model based on the cost function $\mathrm{J}(\theta)$. By using this iterative scheme, the parameters (i.e., the weights and bias term of each neuron in the ANN) are updated, after applying the training set D, such that the cost function is minimized. When the change of the parameter values are small enough that it can be assumed that the parameters have converged and the model is learned.

The learned solution may not be optimal, since the back-propagation algorithm can be trapped in a local minimum. This is due to the high nonlinear nature of the cost function in the parameter space. Often better performance can be obtained by using a pattern-by-pattern mode, also online mode, to learning the model. In this case the weights of the ANN are updated at every time instance a new pattern is presented.

Additionally, it is also recommended to randomize the data sequence prior to using it for learning. In practice it was shown that the pattern-by-pattern mode result in faster convergence and better solutions [7].

4.5 Feedback Control Approach

In the following, we consider a SoS in which one of the CSs is a computing cluster used to run compute-intensive and/or data-intensive business logic of the cyber-physical system. In the following, we first illustrate the impact of environmental changes and system configuration parameters on the performance and availability of such CSs. We then present a possible implementation of the Analysis component of MAPE through behavioural modelling and a possible implementation of the Planning/Execution components of MAPE through feedback control.

MapReduce is a popular programming model and execution environment for developing and executing distributed data-intensive and compute-intensive applications [8]. However, the complexity of configuration of such systems is continuously increasing. Although the framework hides the complexities of parallelism from the users, deploying an efficient MapReduce implementation poses multiple challenges. MapReduce's ad-hoc configuration and provisioning require a high level of expertise to tune [9]. Ensuring performance and dependability of MapReduce systems still poses several challenges.

One of the most popular open source implementations of the MapReduce programming model is Hadoop. It is composed of the Hadoop kernel, the Hadoop Distributed Filesystem (HDFS) and the MapReduce engine. Hadoop's HDFS and MapReduce components originally derived from Google's MapReduce and Google's File System initial papers. HDFS provides the reliable distributed data storage and the MapReduce engine provides the framework to efficiently analyse this data.

In the following, we consider a CS that is a MapReduce cluster that consists of sub-CSs represented by N nodes. A MapReduce workload is defined as the number of concurrent clients (C) that are sending requests to the central controller. Admission control is a classical technique to prevent server thrashing. It consists of limiting the maximum number of clients (MC) that are allowed to concurrently send requests to the central controller.

The performance of MapReduce systems can be measured as the average time (Rt) needed to process a request in a certain time window. Low client response time is a desirable as it reflects a reactive system. The average *Rt* can, for instance, be calculated at every 30 s, using a sliding window with period 15 min.

$$Rt[s] = avg(Rt_1, Rt_2, \ldots, Rt_N) \tag{6}$$

Availability (Av) refers to the accessibility of the system to users. MapReduce is available if the user requests are accepted at the time of their submission. Availability is instantaneous and concentrates on the fraction of time where the system is operational in the sense of being accessible to the end user. Availability is measured as the ratio of accepted MapReduce client requests to the total number of requests, during a period of

time. T here is the previously defined sliding time window size that is used to assign a measurable dynamics to the system. Since T is constant for all experiments, we use only the percentage (%) symbol as the availability measurement unit in all the plots to simplify their understanding.

$$Av\left[\frac{\%}{T}\right] = \frac{N_{SuccessfulJobs}}{N_{SucessfulJobs} + N_{RejectedJobs}} * 100 \tag{7}$$

Furthermore, the service cost is a linear function of the MapReduce cluster size (N), and can be inferred directly from N.

Finally, performance and availability metrics are part of the SLA of the MapReduce system. The SLA specifies MapReduce service level objectives (SLOs) in terms of, for instance, the maximum response time Rt_{max}, and the minimum availability Av_{min} to be guaranteed by the MapReduce system.

Figures 16, 17, 18 show the impact of the variation of, respectively, the workload exogenous variable, the MapReduce cluster size control variable, and the MapReduce cluster's admission control variable on performance and availability metrics. Thus, there is no one-fits-all configuration; rather, a solution that meets a combination of service level objectives as described below.

Example of a Behavioral Model. Capturing the complex behaviour of MapReduce CSs is highly challenging. We propose a model that captures the dynamics of MapReduce CSs, and renders their levels of performance and availability. The model is

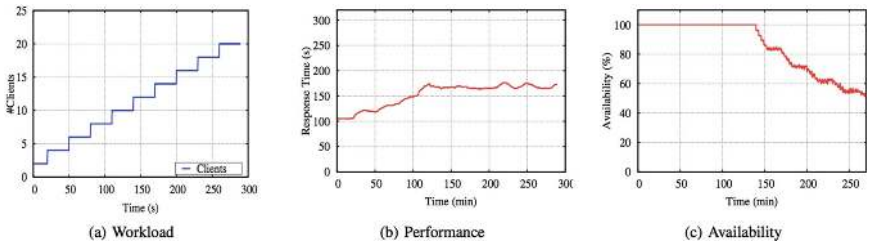


Fig. 16. Impact of workload on MapReduce performance and availability with #Nodes = 20, #MC = 10

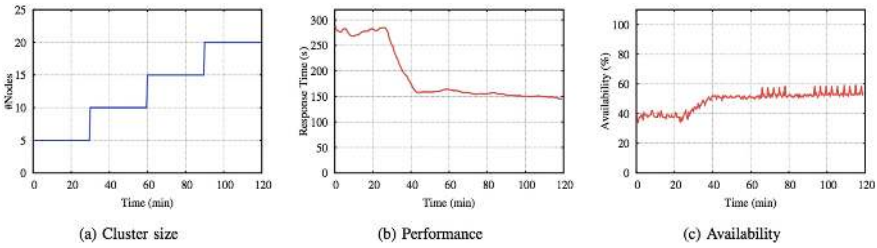
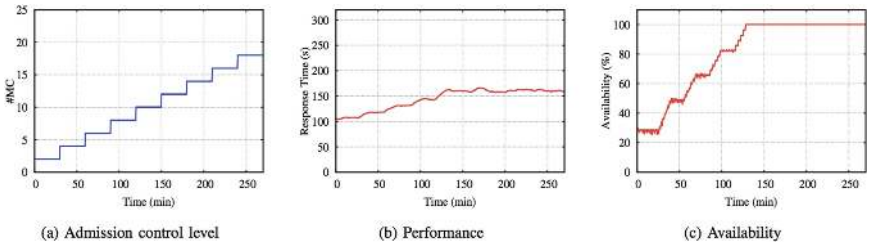


Fig. 17. Impact of cluster size on MapReduce performance and availability with #Clients = 10, #MC = 5



(a) Admission control level (b) Performance (c) Availability

Fig. 18. Impact of admission control on MapReduce performance and availability with #Nodes = 20, #Clients = 10

built as a set of difference equations - as for biological or economical systems - that describe the impact of input variables' variations on system's output variables. We apply a novel modelling approach that considers the MapReduce system as unknown and derives a mathematical model based only on the impact of the input variations on the system's outputs. This technique is part of what we call system identification in control theory. Roughly speaking, one provides known input variation functions (e.g. a step or sinusoidal variation) to the system, and measures the system response to this excitation. Using the output measurements an identification algorithm can approximate the system's internal dynamics. In most cases, without a loss in generality, 1st or 2nd order polynomial difference equations capture the system behaviour sufficiently well.

Figure 19 describes the proposed model variables. The inputs of the model are: exogenous input C that represents the number of clients accessing the underlying MapReduce system, in addition to tunable parameters that can be used to control the MapReduce system, namely the number of nodes N of the underlying MapReduce cluster, and the maximum number of clients MC concurrently admitted in the MapReduce system. In addition to input variables, the model has the following output variables: the average response time Rt to a MapReduce client request, and the level of availability Av of MapReduce to its clients. In the following, we describe the proposed model through the formulas of its output variables.

Example of Feedback Control. A first attempt in controlling the response time of a MapReduce system by adding and removing nodes was realized in [10] by using a PI and a feedforward controller. We design MR − Ctrl, an optimal controller, able to deal

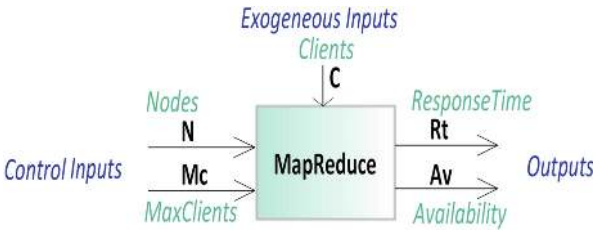


Fig. 19. System model inputs and outputs

with contradictory objectives. As our MapReduce model has two outputs, MR – Ctrl will assure at the same time the response time and the availability specified in the SLA, while minimizing resource utilization.

The complete schema of the control architecture is presented in Fig. 20. All the variables used in the figure are defined in Table 1. More details regarding the implementation of the control framework can be found in [10]. As in Fig. 19, we consider the MapReduce system having two inputs (concatenated in the two dimensional vector u), one exogenous uncontrollable disturbance input C and two outputs (concatenated in the two dimensional vector y). Vector u contains the number of nodes in the cluster N and the max number of clients MC. While the y vector contains the response time Rt and availability Av.

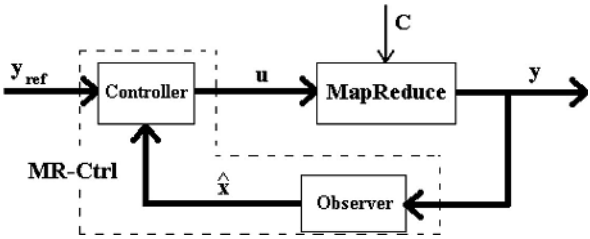


Fig. 20. The control architecture

Table 1. Definition of control variables

$y_{ref} = \begin{pmatrix} Rt_{ref} \\ Av_{ref} \end{pmatrix}$	Reference – response time and availability set in the SLA
$y = \begin{pmatrix} Rt \\ Av \end{pmatrix}$	Measured system output – response time and availability
$u = \begin{pmatrix} N \\ MC \end{pmatrix}$	System control input – number of nodes in the system and the maximum number of clients
C	Disturbance – number of clients trying to connect to the system
$\hat{x}$	Reconstructed behavior of MapReduce

5 Conclusions

This chapter describes the overall approach for managing SoS dynamicity. Its main intent is to associate a Service Level Agreement with SoS, and to provide SLA guarantees in terms of dependability, security, performance, etc. The overall MAPE Monitoring/Analysis/Planning/Execution approach is followed for SoS dynamicity management. The approach is illustrated through different implementations and techniques, e.g., a scalable monitoring, feedback control-based behavioural modelling and reaction strategies.

References

1. AMADEOS, Deliverable D3.1 - Overall Architectural Framework (2015)
2. AMADEOS, Deliverable D4.1 - Case study and use cases (2015)
3. AMADEOS, "Deliverable D4.2 - Case study realization," (2016)
4. Kappen, B.: Stochastic optimal control theory, Lecture Notes, Radboud University Nijmegen (2012)
5. Li, P.Y.: Advanced Control System Design, Ch. 6, Lecture Notes, University of Minnesota (2012)
6. Dreyfus, S.E.: Artificial neural networks, back propagation, and the Kelley-Bryson gradient procedure. J. Guidance Control Dyn. **13**(5), 926–928 (1990)
7. Theodoridis, S.: Machine Learning: A Bayesian Optimisation Perspective. Elsevier, London (2015)
8. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. In: USENIX Symposium on Operating Systems Design and Implementation (OSDI) (2004)
9. Lin, X., Tang, W., Wang, K.: Predator: an experience guided configuration optimizer for Hadoop MapReduce. In: IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom), Taipei, Taiwan (2012)
10. Serrano, D., Bouchenak, S., Marchand, N., Robu, B., Berekmeri, M.: IFAC World Congress (2014)