Proceedings of TMCE 2020, 11-15 May, 2020, Dublin, Ireland, edited by I. Horváth and G. Keenaghan

HOW CAN A SMART CYBER-PHYSICAL SYSTEM VALIDATE ITS RUN-TIME ADAPTATION ACTIONS BEFORE AND AFTER EXECUTING THEM?

Jože Tavčar
Faculty pf Mechanical Engineering
University of Ljubljana
Slovenia
joze.tavcar@lecad.fs.uni-lj.si

Imre Horváth
Industrial Design Engineering
Delft University of Technology
The Netherlands
i.horvath@tudelft.nl

ABSTRACT

The time has come for engineered systems to behave smartly. To this end, they must (i) collect data directly from real-life processes, (ii) build situation awareness, (iii) reason about their operational states, (iv) determine the best servicing objectives, (v) plan their run-time adaptation, and (vi) provide dependable operations/services even under dynamically changing circumstances. Engineers must figure out how to design smart cyber-physical systems (S-CPSs) for adaptation at run-time. Designing such S-CPSs is a challenging task. S-CPSs should not draw up only device run-time adaptation plans, but also confirm their feasibility and efficiency. S-CPSs should predict the physical and computational resources. The theory and methodology behind smart systems is still under development. This paper focuses on run-time adaptation, provides an overview on extant research efforts, and analyses the results published so far. The literature informs about the fact that there is a need for a meta-model of systems' self-adaptation, which might however be completely different depending on the kind of systems and the applications. Therefore, the paper proposes that managing self-adaptation decomposes to four logical stages: (i) planning self-adaptation, (ii) verification before self-adaptation, (iii) operationalization of self-adaptation, and (iv) validation of self-adaptation.

KEYWORDS

Smart cyber-physical systems, self-adaptation, pre-adaptation self-verification, post-adaptation self-validation, design approaches, design principles, system development

1. INTRODUCTION

The paradigm of cyber-physical systems is rapidly evolving [1]. Future cyber-physical systems will exhibit significant differences not only in terms of their functionalities and applications, but also in terms of their implementation and management [2]. They will behave smartly and purposefully, and manage themselves quasi-autonomously [3] [4]. This rapid evolution raises many questions concerning designing for functionality, adaptability, and behavior, as well as about the approaches of dynamically managing complexity, dependability, and reliability [5] [6] [7]. This paper intends to contribute to the specific issue of designing for adaptability, in particular run-time 'design' of self-adaptation [8]. In order to narrow down the domain of discourse, only second-generation CPSs are considered in our study [9].

Designing smart cyber-physical systems (S-CPSs) for adaptation at run-time is a challenging task at least for three reasons. First, S-CPSs should not only devise run-time adaptation plans, but also confirm their feasibility and efficiency [10] [11]. Second, S-CPSs should predict the required physical and computational resources based on service demands and acquire them as needed for self-adaptation at run-time [12] [13] [14]. Third, the underpinning theory and development methodology of socially smart/cognitive systems is still in its infancy [15] [16]. This paper focuses on the first issue, provides an overview of extant research efforts, and analyses the results published so far. The literature informs about the fact that there is the need for a meta-model of systems' self-adaptation, which might however be completely different depending on the kind of systems and the applications [17] [18]. Therefore, the driving hypothesis behind the background research has been

that managing self-adaptation decomposes to four logical (procedural) stages as shown in Figure 1, and will be used as a conceptual reasoning model to be tested in the rest of the paper. We propose referring to this scheme as the generic process of supervised self-adaptation (SSA).

The proposed reasoning model implies the need for studying four specific run-time phenomena of smart CPSs: (i) origination of objective- and situation-sensitive plans and actions for behavioral self-adaptation, (ii) verification of the self-adaptation plans and actions before executing the adaptation, (iii) realization of the intended behavioral self-adaptation, and (iv) validation of the outcome and impacts of self-adaptation after completing the adaptation. SSA is assumed a recurrent system operation that occurs multiple times (with various frequencies) and that may lead to introducing changes in transient operational states, rather than only in steady state system situations.

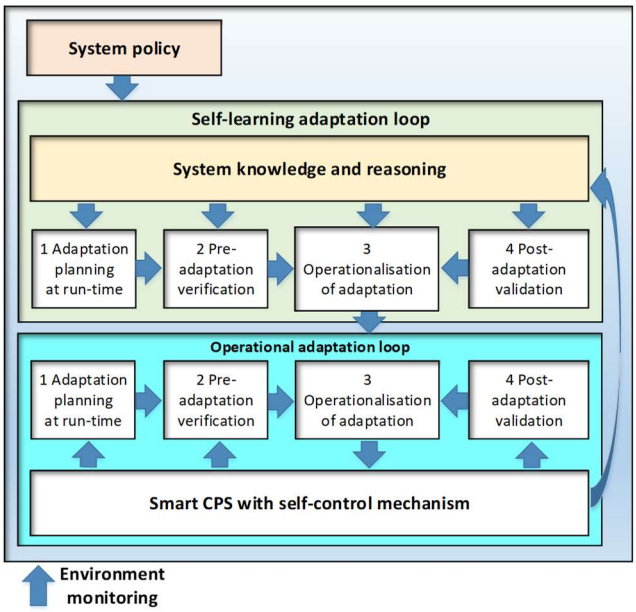

Figure 1 The assumed generic process flow of supervised self-adaptation. Pre-adaptation, self-verification, and post-adaptation self-validation in the smart CPS control loop.

The paper concentrates on two specific research questions, namely: (i) how adaptation plans and actions can be verified before their execution, and (ii) how the results and impacts can be validated after the execution of adaptation. However, in order to be able to answer these context dependent issues, we must first address a somewhat more general research question, namely: (iii) how can S-CPSs adapt themselves at run-time? Although researchers on self-adaptation have established solid principles, such as quiescence, MAPE, meta-requirements, and run-time models, there is currently no comprehensive theory that underpins self-adaptation, is the conclusion that Weyns made [19]. An important challenge that crosscuts the different waves will be developing robust approaches and demonstrating their applicability and value in practice. Essential to that will be the gathering of empirical evidence based on rigorous methods, in particular controlled experiments and case studies [19].

Zheng et al. argued that existing formal methodological techniques and simulation are insufficient for supporting the development of entire general-purpose CPS [20]. The current state of the practice in CPS verification and validation remains an ad-hoc trial and error process. There are still significant gaps between the formal models of computing and the formal models of physics that underpin today's CPS systems.

1.2 Self-verification and self-validation

In the above sub-section, three concepts have been introduced: (i) run-time self-adaptation, (ii) self-verification, and (iii) self-validation. They seem to be important for general-purpose CPSs, and even more for smart CPSs with the ability of SSA. Though these terms are not brand new, several different interpretations and operationalization are present in the literature [20]. To reduce ambiguity, below we elaborate on these terms and put them in the context of this study.

Though the terms 'verification' and 'validation' frequently occur in system science and engineering literature, 'self-verification' and 'self-validation' seem to be novel and under-elaborated concepts. Verification and validation are two representative approaches to justification, which may be based on rational and empirical, and direct or indirect methods. Justification strives after logical consistence, semantic coherence, factual reliability, etc. The question driving justification is whether the theoretical fundamentals, conceptual framework, working principle, and knowledge constituents provide sufficient underpinning and/or framing for the purpose. Verification generates indicators for consistency, coherence, and reliability. In engineering, a rather pragmatic interpretation of verification is testing against requirements.

While verification refers to internal properness, validation refers to appropriateness. In the tradition of scientific inquiry, validation is a multi-faceted activity

focusing on the confirmation of knowledge. Validation is used to test and prove appropriateness in a different context, for a larger population, under specific conditions, as well as utility for a purpose in some (application) context. In engineering, validation is testing the fulfilment of the expected functionality.

With a view to their interconnected roles in demonstrating properness and appropriateness, verification and validation are inseparable. Likewise, self-verification and self-validation play interrelated roles in adaptation testing of CPSs. Self-V&V has been envisaged as a repertoire of system abilities that enables the system capable to eventually design and complete all necessary verification and validation tasks that are needed to ensure its dependable and optimal operation [21]. Traditionally, systems for are planned for adaptation in the functional design and/or architectural design phases. Self-adaptable systems, however, must be 'designed' by the system itself at run-time (or, in certain specific cases, with the assistance of system operators) [22]. As methods of verification of embedded systems, the authors of the above paper identified: (i) simulative verification, (ii) emulative verification, and (iii) formal verification. The run-time self-adaptation of smart cyber-physical systems is still largely terra incognita, lacking comprehensive theoretical and methodological underpinning. A number of papers discuss the various forms of verification and validation of self-tuning (e.g., self-resilience), but much less the forms of self-adaptation (structural, functional, behavioral). The self-evolution of complicated engineered systems has been recognized as an intellectual and technological challenge, but relatively scarcely addressed and only in non-generic cases from a methodological and technical perspectives [23] [24].

2. SELF-ADAPTATION OF SMART CYBER-PHYSICAL SYSTEMS

2.1. Previous work on theories and principles of self-adaptation

The literature discusses many principles of self-adaption depending on how it is operationalized within the system. The principles opposing each other define some sort of dimension, in which specific solutions can be generated. The dimensions of the juxtaposition principles are shown in Table 1.

The major issue is that the system should continue its operation and servicing while also adapting in its functionality and architecture at run-time. This means that not only the introduction of the changes, but also maintaining operational continuity is a concern, though sometimes it is not explicitly considered. For instance, Zhou, P. et al. differentiated between two types of self-adaptation of CPS: (i) environment-centered self-adaptation (for a proper interaction with changing environment) and (ii) system-centered adaptation (guaranteeing the dependability of the cyber space) [6].

Table 1: Juxtaposing the major aspects of self-adaptation

aspect of self-adaptation	on the one hand	on the other hand
computational enabler	model-based	data-driven
organizational construct	centralized	decentralized
intensity of agency	proactive	reactive
criterion for execution	performance-based	task-based
resources concerned	hardware	software and cyberware

The efforts for making self-adaptation an engineering reality got underway more than a decade ago. Krupitzer, C. et al. published an overview of the engineering approaches to self-adaptive systems. They identified various characteristics of self-adaptation (such as reason, level, time, technique, and control) and developed a taxonomy based on these. In addition, they conceptualized two generic constituents of adaptive systems, namely, the adaptation logic (AL) and the managed resources (hardware, software and cyberware) [26]. The AL executes the MAPE-RL functions (i.e. monitors the environment, analyses the data for change, plans adaptation, controls the execution of the adaptation, reasons in context, and learns from adaptations).

In his paper overviewing the recent developments, Weyns discussed the fact that, though researchers have established solid principles in the field of self-adaptation, such as quiescence, MAPE, meta-requirements, and run-time models, there is still no comprehensive theory that could underpin the computational mechanisms of the self-adaptation of complicated systems [19]. He explained the current situation as an outcome of the different directions of research and development. An important challenge that crosscuts the different directions will be to develop robust approaches and demonstrate their applicability and value in practice. Essential to that

will be the gathering of empirical evidence based on rigorous methods, in particular controlled experiments and case studies [19].

Pradhan et al. described the work on improving run-time support for autonomous resilience via self-reconfiguration [28]. Run-time infrastructure governs self-reconfiguration mechanisms. The application of adaptation methods begins with setting goals.

2.2. Model-based self-adaptation versus data driven self-adaptation

One issue of self-adaptation is to derive metrics criteria for and make decisions on adaptation. A typical method of capturing the criteria and making decisions is using system models that describe the 'as-is' and the 'to-be' states. Other alternative is executing policies and/or reasoning with objectives. Adaptation may also be controlled by run-time acquired operational or servicing data. Model-based adaptation control must cope with operational dynamics that may arise as a result of changes in system states or environment states. Reasoning in context and learning from the adaptation functions of MAPE-RL are seen as options supporting dynamic functional, architectural, and behavioral model management at run-time.

In the case of model-based operation control, the first issue is validating the system model itself. McCarl discussed multiple issues in validating models and proposed that model validation relates to its application [29]. He also proposed considering three categories of models, such as (i) exploratory models (examining how phenomena enter into the formation of reality), (ii) predictive models (forecasting the consequences of decisions), and (iii) prescriptive models (involving the solution of a specific decision maker problem). These types of models can be correlated with models that are needed for smart-system operation.

Incki and Ari elaborated on the utilization of a model-based run-time monitoring approach for providing reliable service. Message sequence charts are used, later allowing practitioners to express an application's expected behavior in terms of complex-event processing patterns [30]. The presented approach enables the non-intrusive monitoring of IoT behavior at run-time. Perrouin et al. discussed that adaptive CPS's ability to evolve is limited to the addition, update, or removal of adaptation rules or reconfiguration scripts [31]. They suggest leveraging recent advances in model-driven techniques to offer an approach that supports the evolution of both systems and their adaptation capabilities. The basic idea is to consider the control loop itself as an adaptive system. García-Valls et al. proposed a solution for designing adaptive cyber-physical systems by using parametric models that are verified as the system operates, so that adaptation decisions are made based on the timing requirements of each particular adaptation event [32]. Their approach allows the system to make timely adaptations that exploit the potential parallelism of the software and its execution over multicore processors.

2.3. Centralized self-adaptation versus decentralized self-adaptation

Self-adaptation can happen according to a top-down or a bottom-up control strategy. This results in a fully centralized adaptation planning in the former case, and in a fully decentralized adaptation planning in the latter. In practice, various combinations of the top-down and the bottom-up control strategies are applied. For centralized self-adaptation, a selected arbiter collects all the necessary data and knowledge from the system components, solves the satisfiability (SAT) problem, and reliably communicates the result back to the components. Gerostathopoulos et al. identified two forms of decentralized self-adaptation (DSA), namely: (i). DSA with distributed consensus, and (ii) DSA with no distributed consensus [33]. In the first case, each component solves the SAT problem locally, based on its local knowledge (local view of system state). The results are then communicated to and agreed upon between all components. In the second case, each component solves SAT problem locally based on its local knowledge, (local view of system state), without requiring this knowledge to be synchronized across nodes. This allows components even to keep their autonomy detached from the network [33]. A decentralized self-adaptation mechanism for service-based applications was proposed by Nallur & Bahsoon [34].

2.4. Proactive self-adaptation versus reactive self-adaptation

From the perspective of control theory, open- and closed-loop controlled systems can be differentiated. Closed-loop controlled systems adapt themselves based on the deviations observed in the output parameters, and they are monitored either by feedforward or feedback mechanisms. This type is called reactive self-adaptation. Open-loop systems monitor changes to input parameters in combination

with (internal) operational parameters and adapt operation based on forecasting the expected changes in the output variables. This type is proactive self-adaptation, which becomes complicated under uncertainties concerning the input parameters and the operational environment. The aim of proactive adaptation is adapting before it becomes necessary, it is based on prediction.

2.5. Hardware self-adaptation versus software/ cyberware self-adaptation

The self-adaptation of the physical constituents and the architecture as a whole of the system play a crucial role in the case of each of the three sub-categories of polymorphic systems. The action plan should take into consideration the continuing operation of the hardware components. Thus, adaptation plan generation starts with creation of adaptation requests, based on the relevant system states, events, and situations.

Braberman et al. propose the MORPH reference architecture, which allows both independent reconfiguration and behavior adaptation [35]. The architecture is structured in three main layers that sit above the target system: Goal Management, Strategy Management and Strategy Enactment. Orthogonal to the three layers is the Common Knowledge Repository. Each layer can be thought of as a implementing a MAPE-K loop.

2.6. Main findings concerning the principles of self-adaptation

Existing mainstream model checking techniques and tools were not conceived for run-time usage; hence they hardly meet the constraints imposed by on-the-fly analysis in terms of execution time and memory usage [36] [37].

According to Gerostathopoulos et al., failures may appear when a self-adaptive CPS operates under environment condition, for which they are not specifically designed [33]. The homeostasis of CPS was improved by introducing run-time changes to the architecture-based self-adaptation strategies according to environment stimuli. Four mechanisms that reify the idea were introduced: (i) collaborative sensing, (ii) faulty component isolation from adaptation, (iii) enhancing mode switching, and (iv) adjusting guards in mode switching.

Conclusions from Muccini et al. research on self-adaptation: 64% of the studies apply adaptation at the application layer and 24% at the middleware layer. MAPE (Monitor-Analyse-Plan-Execute) is the dominant adaptation mechanism (60%), followed by agents and self-organization (both 29%) [38]. The findings show that adaptation in CPS is a cross-layer concern, and the promising solutions combine various adaptation mechanisms within and across layers.

Klös, et al. extended the MAPE-K feedback loop architecture by imposing a structure and requirements on the knowledge base and by introducing a meta-adaptation layer [3]. This enables the continuous evaluation of the accuracy of previous adaptations, learn new adaptation rules based on executable run-time models, and verify the correctness of the adaptation logic in the current system context.

3. ADAPTATION PLAN VERIFICATION BY SMART CYBER-PHYSICAL SYSTEMS

3.1. Objectives of pre-adaptation self-verification

When the adaptation plans and action specifications are available, their properness should be checked before they are executed by the system. This verification should (i) happen with minimal or zero external influence, (ii) be completed in a time frame determined by the system operation, (iii) be realized by the system without restricting the actual operation, and (iv) feed back to the adaptation planning activities.

3.2. Possibilities for pre-adaptation self-verification

Bianculli et al. introduced a general framework, called Syntax-DrivEn inCrementAl veRification-SiDECAR, for the definition of verification procedures, which are made incremental by the framework itself [39]. Verification procedures are driven by the syntactic structure of the system and encoded as semantic attributes associated with the grammar. Incrementalism is achieved by coupling the evaluation of semantic attributes with an incremental parsing technique.

Ghezzi et al. argued that there is still a serious mismatch between verification and modern development processes, which focus strongly on agility and incremental, iterative development [40]. Verification must become agile, and seamless introductions into agile processes must become feasible. Verification-driven development and agile verification can be achieved. Jiang et al. conducted a

study on applying run-time verification to cooperate with current decision support system (DSS) based on real-time data [41].

A run-time verification technique is proposed and formalized to strengthen the medical DSS. It combines formal methods in software engineering and practice guidelines in medicine to rigorously verify run-time temporal properties automatically.

Pinisetty et al. introduced a method for predictive run-time verification of timed properties, where the system is not entirely a black-box, but something about its behavior is known a priori. A priori knowledge about the system's behavior allows the verification monitor to foresee the satisfaction (or violation) of the monitored property [42]. In addition to providing a conclusive verdict sooner, the verification monitor also provides additional information such as the minimum (maximum) time when the property can be violated (satisfied) in the future.

Bauer et al. presented a run-time verification approach for properties expressed either in linear time temporal logic (LTL) or timed linear time temporal logic (TLTL), suitable for monitoring discrete-time and real-time systems, respectively [43]. For LTL, a conceptually simple monitor generation procedure is given, which is optimal in two respects: First, the size of the generated deterministic monitor is minimal, and, second, the monitor identifies a continuously monitored trace as either satisfying or falsifying a property as early as possible.

In order to prevent the occurrence of undesired behavioral adaptations, a run-time correctness verification approach was introduced by Cardozo et al. This approach uses a symbolic execution engine to reason about the system's reachable states whenever contexts are activated or deactivated [10]. Context activation and deactivation requests are allowed depending on the presence of erroneous states within reachable states. Calinescu et al. discussed the potential and challenges associated with the run-time use of quantitative verification and model checking as a way of obtaining dependable self-adaptive software [44].

The main contribution of Filieri et al. is the description of a mathematical framework for run-time efficient probabilistic model checking. The approach statically generates a set of verification conditions that can be efficiently evaluated at run-time as soon as changes occur [37]. The proposed approach also supports sensitivity analysis, which enables reasoning about the effects of changes and can drive effective adaptation strategies. Their paper addresses this issue and focuses on the perpetual satisfaction of non-functional requirements, such as reliability or energy consumption. Its main contribution is the description of a mathematical framework for checking run-time efficient probabilistic models. Our approach statically generates a set of verification conditions that can be efficiently evaluated at run-time as soon as changes occur. The proposed approach also supports sensitivity analysis, which enables reasoning about the effects of changes and can drive effective adaptation strategies

3.3. White spots and open issues

Though the idea of run-time verification is already addressed from many perspectives in the literature, it seems that it is used for largely different purposes and in largely different manners. In certain cases it does not serve as verification of the adaptation before it occurs, but as a means to monitor the execution of adaptation in run-time. There are cases when it is used for the purpose of validation of the results of run-time adaptation. For example, Mitsch and Platzer introduced ModelPlex, which provides correctness guarantees for CPS executions at run-time [45]. It combines the offline verification of CPS models with run-time validation of system executions for compliance with the model. ModelPlex ensures in a provably correct way that the verification results obtained for the model apply to the actual system runs by monitoring the behavior of the world for compliance with the model. An additional contribution is a systematic technique of automatically synthesizing provably correct monitors from CPS proofs in differential dynamic logic by a correct-by-construction approach, leading to verifiably correct run-time model validation.

Zheng et al. argued that existing formal method and techniques simulation are insufficient for supporting the development of entire general-purpose CPS [20]. The current state of practice in CPS verification and validation remains an ad-hoc 'trial and error' process. There are still significant gaps between the formal models of computing and the formal models of physics that underpin today's CPS systems.

4. PROPOSAL FOR A COMPREHENSIVE SELF-ADAPTATION STRATEGY FOR SMART CYBER-PHYSICAL SYSTEMS

4.1. The overall reasoning model of

verification and validation of self-adaptation at run-time

Run-time verification and validation is split into four phases for easier understanding as shown in Figure 1. However, the listed activities are interconnected and they are conducted as a constant information flow. Run-time validation must change our mindset. We must switch from traditional validation in the design phase, where the most critical situations can be tested case by case into the run-time domain, where permanent monitoring and adaptations enable control in several small steps.

Operational adaptation in the first loop and system control mechanism adaptation in the second control loop must be distinguished, as presented in Figure 1. The first one responds to environment changes or adapts the system to new goals. The adaptation in the second loop updates the system – putting the new logic into the system.

4.2. Adaptation planning at run-time

Figure 2 presents subtasks at the point of adaptation planning. It is not a one-time activity, but a set of non-stop executing activities. Environment and system parameters are monitored at run-time. During system design or later at system upgrades (second control loop), the range of allowed parameter value is defined. If the value of specific parameters is out of the normal range, the system is switched to safe mode. Smart CPSs are of different configurations and complexity. For simplicity of understanding, we explain the situation in environment or in system with parameters.

The third input into adaptation planning is a request for adaptation. A trigger for adaptation can be: (i) a request from the first control loop caused by environment changes, (ii) a new system goal is defined, (iii) the system configuration or reasoning logic is updated. The input for adaptation is defined at a high level – it does not include all the details of settings that are necessary for achieving the goal. Adaptation planning considers in real time the system state (environment and system parameters), system policy, and system knowledge. The adaptation proposal must be feasible for adaptation in the target system. The result is an adaptation plan that contains all detailed system settings needed to achieve target adaptation.

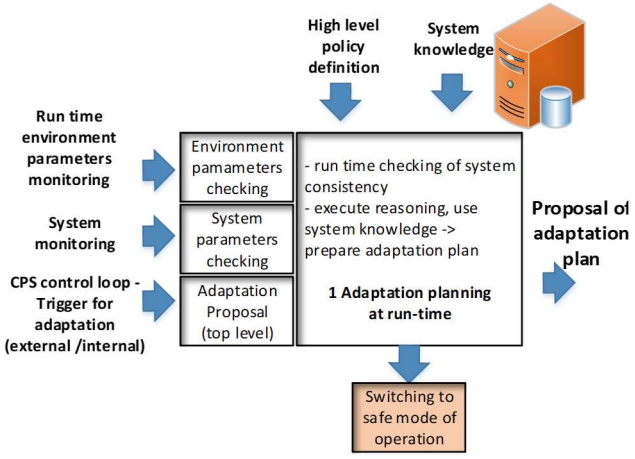

Figure 2 Adaptation planning at run-time

4.3. Pre-adaptation self-verification

Operations are adapted at the first level of the control loop within the range of predefined limits. Data from the adaptation plan, along with environment and system parameters, are inserted into the system model. If the results of the model simulation are consistent with the adaptation proposal and system policy, the adaptation plan is approved. Smart CPS could have specific verification methods (Figure 3).

It must again be stressed that adaptation planning and pre-adaptation verification is a continuous process. Adaptation occurs in small incremental steps that assure a smooth and controlled transition from one state to new one.

A special case of adaptation is adaptation at the system level, which means the system control mechanism or system configuration is changed. Such adaptation must be done with special care, while new control rules can significantly change system behavior.

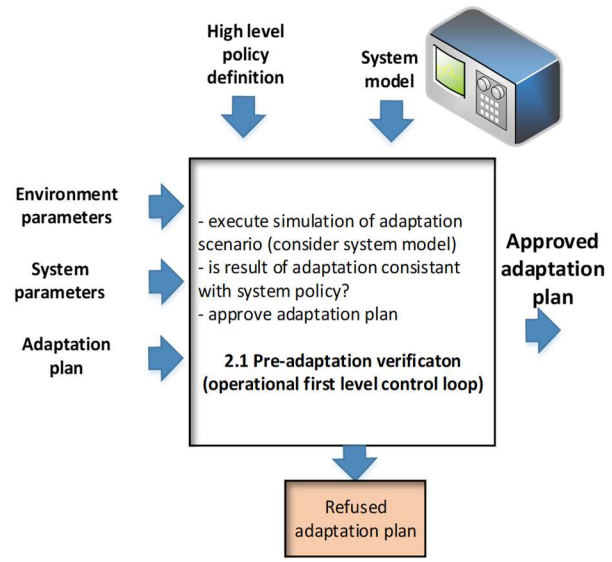

Figure 3 Pre-adaptation self-verification – first level of control loop

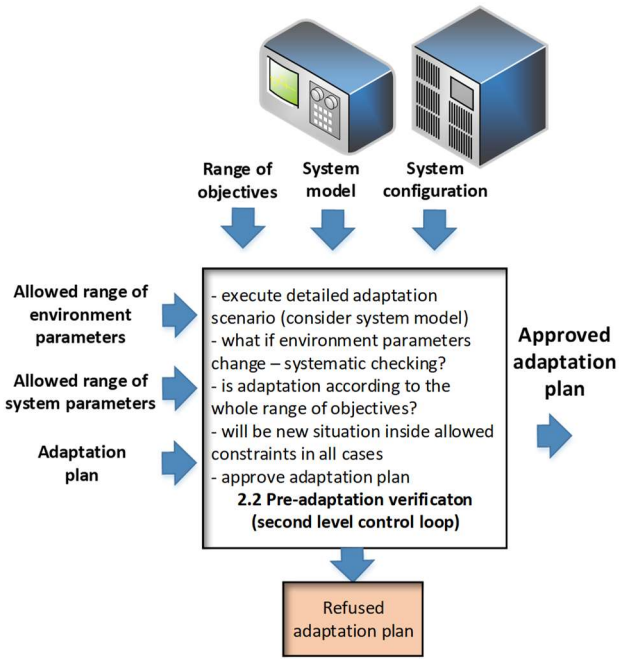

Figure 4 Pre-adaptation verification - second level of control loop

Option 1: Systemic checking of new system is needed before adaptation implementation at the second control loop. This verification is in some ways similar to that in the design phases of product design. Critical situations are checked for the allowed range of environment and system parameters, which must be done in the virtual world. The whole procedure can take a long time, wherefore verification must be conducted in several steps and in the idle periods when computation capacity is available.

Option 2: Is radical system adaptation and systemic checking in the whole range of parameters realistic? How can we check models if the system model is changed? It must always be adapted in incremental steps.

For example: a new rule is added; all the previous rules are still valid and checking must only be focused on the new rule. The incremental approach enables verification of all small adaptations with all four phases of adaptation. The adaptation loop is at the same time a learning loop, which in real time accepts or refuses small changes at the second level of the control loop. That also means the allowed range of system and environment parameters can be incrementally extended.

4.4. Operationalization of self-adaptation

Self-adaptation is conducted according to the approved adaptation plan. However, adaptation and system consistency checking as presented in Figures 3 and 4 are running concurrently. That means there are two control levels; the first one is the pre-adaptation verification and the second one is run-time monitoring, and immediate switching of the system back or into safe mode of operation if the system parameters run out of allowed limits.

A realistic scenario includes smooth system adaptation, but later changes of environment parameters can cause the system into unpredicted situation according to the existing model. Run-time monitoring and immediate reaction are permanent safeguards that assure reliable system operation.

4.5. Post-adaptation self-validation

The main objective of post-adaptation self-validation is comparison between predicted model in pre-adaptation and current system situation after adaptation. Comparison results in valuable data for the self-learning process. If the difference is small, the system model is additionally confirmed. In the opposite case, the system model is updated with a new adaptation example (Figure 5).

Run-time monitoring assures safe operation and enables the conduct of a learning loop even with parameters values that have not been tested during the system design phase. However, system evolution must be done in several incremental steps. A larger number of adaptation cycles with a predefined range of parameters value improves the system's knowledge and consequently improves the prediction of system behavior. With several small incremental steps, the

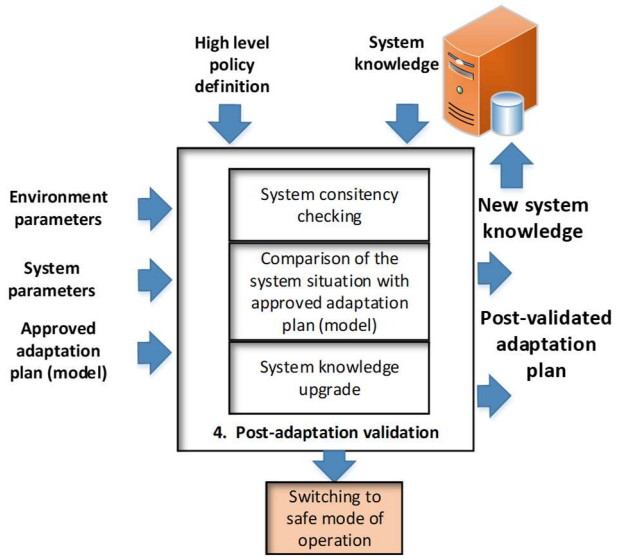

Figure 5 Post-adaptation self-validation

range of the system's operation, structure, and configuration can be expanded into a new mode of operation that was unknown at the time of design. Most examples from the literature refer to self-tuning – the first level of control loop.

4.6. A demonstrative case study: Run-time self-adaptation of a robotic vacuum cleaner

Robot cleaners are currently quite popular applications. Many people have already personal experience with its usage. Therefore, it is used as a case study of adaptation phases.

A) A robot cleaner without self-learning capability

A robot has rules for operation that were defined already in the design phase. It uses the same strategy all the time and in any room. Rules for operation:

- If a barrier or a curb is detected in front of the robot, go back by 50 cm, and turn right with a random angle value between 30 and 60 degrees. Go forward.
- If there is only 5% of energy in battery, go to the docking station.
- If the robot is blocked (it cannot move further on), it generates a warning sound and stops operating.

It is assumed that during long enough operation the robot will clean all the room by randomly moving around. Run-time monitoring is used to detect and avoid barriers. Strategy of the robot is only pre-defined reaction to typical situation in environment. We do not classify this kind of robot as smart CPS. According to the 4 phases recognized, there is no adaptation planning and no pre-adaptation verification. Post-validation is limited to run-time monitoring and responses to the data from sensors.

B) A robot cleaner with self-learning capability

A self-learning robot cleaner has advanced room awareness, reasoning, and cleaning path optimization. Environment monitoring is permanently active that assures robust and safe operation as described in section A.

1. Adaptation planning

If the robot starts cleaning in a new, unknown room, it first goes in a randomly selected direction. Adaptation planning in the first control loop means cleaning path planning by considering the situation in the room – what has already been cleaned, where are the barriers located, what is the room configuration, and what is a useful cleaning strategy? If the smart robot can recognize layouts from a previously cleaned room, the learning curve is faster and cleaning more optimal.

2. Pre-adaptation verification

Smart robots do not constitute a safety-critical case. The phases of adaptation planning and pre-adaptation verification could be joined into a single phase. The system collects data from actuators and sensors, and then checks whether the hardware is ready and whether there are barriers in the planned movement path. The robot's parameters must also be consistent with system policy.

3. Operationalization of adaptation

Robot acceleration, movement, rotation, and cleaning happens according to the adaptation plan. Moving parameters enable safe stops in the event of barriers and switching to safe mode operation in unmanageable situations.

4. Post-adaptation validation

During post-adaptation validation, the room map (environment model) is compared with the measured one. New knowledge on the room configuration and what has been already cleaned is stored in a system database. New system knowledge is used for adaptation planning in the next step.

4.7. The implemented learning loop

Smart robot cleaners are actually not terribly demanding applications. In most cases they operate only for an hour per day, meaning that there is enough idle time for system upgrades – second control loop. It could be am upgrade to system policy (for example: maximum moving speed) or to the path generation algorithm or even battery management. For run-time adaptations, small incremental reasoning improvements are only realistic if they are the result of system operation analysis over a longer period of time. That means a reasoning algorithm is upgraded with a single additional rule. The effects of the adaptation can be assessed in most cases only on the basis of longer (several hours) device operation. If the upgrade is assessed well, it is kept in the reasoning algorithm; otherwise it is deleted.

Let us summarize the run-time adaptation approach: adaptation steps for first-level or second-level (learning loop) control loops must be incremental. This means that run-time monitoring and control measure shall have the capability to react in real time

and avoid safety-critical situations. System capability to operate in at least partly "unknown" circumstances open the possibility of operating reliably in changing environments, as well as enabling the self-learning loop and, as a consequence, system self-evolution.

5. DISCUSSION

The paper's title was formulated as a question. This was done to indicate the authors' explorative study. Now the question is whether or not it is possible to answer the question with a high degree of certainty. Three research questions guided the presented study: (i) what adaptation objectives are realistic for S-CPSs and how they can adapt themselves in run-time, (ii) in what forms can adaptation plans and actions manifest and according to which principles can they be verified before execution, and (iii) how can the results and impacts of adaptation be validated after operationalization and after multiple variant system adaptations. One trivial finding of this paper is that run-time self-adaptation is a complicated matter, which should be approached in a reductionist manner.

Nevertheless, self-adaptation is seen as a complex of four computational activities, namely: (i) planning self-adaptation, (ii) verification of plan, (iii) execution of self-adaptation, and (iv) validation of the outcomes. As shown by the literature, there are many proposals for completion of the four activities, but there was no integral methodology proposed yet to embrace all of them.

The proposed model towards run-time adaptation and validation is based on incremental changes of parameters and run-time monitoring. That means modification is possible even outside the range of predefined parameters, while the step backwards is always possible. The approach works well if the system behaves according to linear low. In the case, that specific parameter setting can cause catastrophic situation and even switching to safe mode of operation cannot stabilize the system, the proposed approach cannot be applied. Such situations must be avoided by the allowed range of system parameters and the system policy.

6. CONCLUSIONS

Smart CPSs are purpose-driven and context-dependent behavior and reasoning that make architectural and functional adaptation and evolution possible. Smart CPSs cannot be validated with conventional deterministic approaches. The contribution of this paper manifests in a comprehensive overview of the state of the art and a working proposal towards run-time adaptation and validation in four steps: (i) planning self-adaptation, (ii) verification of plan, (iii) execution of self-adaptation, and (iv) validation of the outcomes. Validation was recognized as an open issue already in the first generation of CPSs. A large number of states cannot be checked in an acceptable time and cost frame. Each realistic technical system operates in unpredictable environment conditions. One solution is run-time adaptation and validation. The paper's contribution works towards the design of smart cyber-physical systems (S-CPSs) for adaptation at run-time. The proposed model contains real-time data collecting and building situational awareness. Run-time adaptation planning is based on reasoning about the system's operational states and provides dependable operations/services even under dynamically changing circumstances.

The survey has shown a gap in advanced validation methods, especially run-time validation, which could be applied for smart CPSs. New methods and approaches have been recognized that enable validation of a higher level of adaptability, system and environment awareness, self-control of constraints and resources in real-time, and run-time validation. The proposed model shows the direction in which additional research must be conducted. The principals for self-constraining must be developed and later approved in practice.

ACKNOWLEDGMENTS

This work was supported by Slovenian Research Agency – ARRS, grant number contract no. P2-0265.

REFERENCES

[1] Krämer, B.J. (2014). Evolution of cyber-physical systems: A brief review. In: Applied Cyber-Physical Systems. Springer, New York, NY. pp. 1-3.

[2] Horváth, I., Rusák, Z., & Li, Y. (2017). Order beyond chaos: Introducing the notion of generation to characterize the continuously evolving implementations of cyber-physical systems. In: Proceedings of the ASME 2017 International Design Engineering Technical Conferences. ASME, pp. 1-14.

[3] Klös, V., Göthel, T., & Glesner, S. (2018). Runtime management and quantitative evaluation of changing system goals in complex autonomous systems. Journal of Systems and Software, Vol. 144, pp. 314-327.

[4] Zambonelli, F., Bicocchi, N., Cabri, G., Leonardi, L., & Puviani, M. (2011). On self-adaptation, self-

expression, and self-awareness in autonomic service component ensembles. In: Proceedings of the 5th Conference on Self-Adaptive and Self-Organizing Systems Workshops, IEEE, pp. 108-113.

[5] Kramer, J., & Magee, J. (2007). Self-managed systems: An architectural challenge. In: Proceedings of the Future of Software Engineering Conference. IEEE Computer Society, pp. 259-268.

[6] Zhou, P., Zuo, D., Hou, K.M., Zhang, Z., Dong, J., Li, J., & Zhou, H.A. (2019). Comprehensive technological survey on the dependable self-management CPS: From self-adaptive architecture to self-management strategies. Sensors, Vol. 19, No. 5, 1033, pp. 1-58.

[7] Dai, W., Dubinin, V.N., Christensen, J.H., Vyatkin, V., & Guan, X. (2017). Toward self-manageable and adaptive industrial cyber-physical systems with knowledge-driven autonomic service management. IEEE Transactions on Industrial Informatics, Vol. 13, No. 2, pp. 725-736.

[8] Gerostathopoulos, I., Bures, T., Hnetynka, P., Keznikl, J., Kit, M., Plasil, F., & Plouzeau, N. (2016). Self-adaptation in software-intensive cyber-physical systems: From system goals to architecture configurations. Journal of Systems and Software, Vol. 122, pp. 378-397.

[9] Tavčar J. and Horváth I. (2019). A Review of the Principles of Designing Smart Cyber-Physical Systems for Run-Time Adaptation: Learned Lessons and Open Issues, IEEE Transactions on systems, Man, and Cybernetics, Systems. Vol. 49, No. 1, pp. 145–158.

[10] Cardozo, N., Christophe, L., De Roover, C., & De Meuter, W. (2014). Run-time validation of behavioral adaptations. In: Proceedings of 6th International Workshop on Context-Oriented Programming, ACM, pp. 1-5.

[11] Horváth, I., Suárez Rivero, J.P., & Hernández Castellano, P.M. (2019). Editorial: Past, present and future of behaviourally adaptive engineered systems. Journal of Integrated Design & Process Science, Vol. 23, No. 1, pp. 1-15.

[12] Spinner, S., Casale, G., Brosig, F., & Kounev, S. (2015). Evaluating approaches to resource demand estimation. Performance Evaluation, Vol. 92, pp. 51-71.

[13] Nayak, A., Reyes Levalle, R., Lee, S., & Nof, S.Y. (2016). Resource sharing in cyber-physical systems: modelling framework and case studies. International Journal of Production Research, Vol. 54, No. 23, pp. 6969-6983.

[14] Huber, N., Brosig, F., & Kounev, S. (2011). Model-based self-adaptive resource allocation in virtualized environments. In: Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, ACM, pp. 90-99.

[15] Bordel, B., Alcarria, R., Pérez-Jiménez, M., Robles, T., Martín, D., & de Rivera, D.S. (2015). Building smart adaptable cyber-physical systems: Definitions, classification and elements. In: Intern. Conf. on Ubiquitous Computing and Ambient Intelligence. Springer, Cham. pp. 144-149.

[16] Jones, K.H. (2014). Engineering antifragile systems: A change in design philosophy. Procedia computer science, Vol. 32, pp. 870-875.

[17] Sabatucci, L., Seidita, V., & Cossentino, M. (2018). The four types of self-adaptive systems: A metamodel. In: Proceedings of the Inter. Conf. on Intelligent Interactive Multimedia Systems and Services. Springer, Cham. pp. 440-450.

[18] Huber, N., van Hoorn, A., Koziolek, A., Brosig, F., & Kounev, S. (2014). Modeling run-time adaptation at the system architecture level in dynamic service-oriented environments. Service Oriented Computing and Applications, Vol. 8, No. 1, pp. 73-89.

[19] Weyns, D. (2017). Software engineering of self-adaptive systems: An organised tour and future challenges. Handbook of Software Engineering. Springer.

[20] Zheng, X., Julien, C., Kim, M., & Khurshid, S. (2017). Perceptions on the state of the art in verification and validation in cyber-physical systems, IEEE Systems Journal, Vol. 11, No. 4, pp. 2614-2627.

[21] Drechsler, R., Fränzle, M., & Wille, R. (2015). Envisioning self-verification of electronic systems. In: Proceedings of the 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip, IEEE, pp. 1-6.

[22] Musil, A., Musil, J., Weyns, D., Bures, T., Muccini, H., & Sharaf, M. (2017). Patterns for self-adaptation in cyber-physical systems. In: Multi-disciplinary engineering for cyber-physical production systems, Springer, Cham, pp. 331-368.

[23] Raichman, N., Gabay, T., Katsir, Y., Shapira, Y., & Ben-Jacob, E. (2004). Engineered self-organization in natural and man-made systems. In: Continuum Models and Discrete Systems. NATO Science Series II, vol 158. Springer, Dordrecht, pp. 187-205.

[24] Shin, M., Mun, J., & Jung, M. (2009). Self-evolution framework of manufacturing systems based on fractal organization. Computers & Industrial Engineering, Vol. 56, No. 3, pp. 1029-1039.

[26] Krupitzer, C., Roth, F.M., VanSyckel, S., Schiele, G., & Becker, C. (2015). A survey on engineering approaches for self-adaptive systems. Pervasive and

Mobile Computing, Vol. 17, pp. 184-206.

[28] Pradhan, S., Dubey, A., Levendovszky, T., Kumar, P.S., Emfinger, W.A., Balasubramanian, D., Otte, W. & Karsai, G. (2016). Achieving resilience in distributed software systems via self-reconfiguration, The Journal of Systems and Software, Vol. 122, pp. 344-363.

[29] McCarl, B. A. (1984). Model Validation: An Overview with some Emphasis on Risk Models. Review of Marketing and Agricultural Economics, Vol. 52, No. 3, pp. 1-21.

[30] Incki, K., and Ari, I. (2018). Model-Based Runtime Monitoring of Smart City Systems, Procedia Computer Science, Vol. 134, pp. 75-82.

[31] Perrouin, G.M., Brice Chauvel, F., Fleurey, F., Klein, J., Le Traon, Y., Barais, O., & Jezequel, J.-M. (2012). Towards flexible evolution of dynamically adaptive systems. In: Proceedings of the 34th Intern. Conference on Software Engineering, Zurich, pp. 1353-1356.

[32] García-Valls, M., Perez-Palacin, D., & Mirandola, R. (2018). Pragmatic cyber physical systems design based on parametric models. Journal of Systems and Software, Vol. 144, pp. 559-572.

[33] Gerostathopoulos, I., Skoda, D., Plasil, F., Bures, T., & Knauss, A. (2019). Tuning self-adaptation in cyber-physical systems through architectural homeostasis. Journal of Systems and Software, Vol. 148, pp. 37-55.

[34] Nallur, V., & Bahsoon, R. (2013). A decentralized self-adaptation mechanism for service-based applications in the cloud. IEEE Transactions on Software Engineering, Vol. 39, No. 5, 591-612.

[35] Braberman, V., D'Ippolito, N., Kramer, J., Sykes, D., & Uchitel, S. (2015). Morph: A reference architecture for configuration and behaviour self-adaptation. In: Proceedings of the 1st International Workshop on Control Theory for Software Engineering, ACM. pp. 9-16.

[36] Tavčar, J., Duhovnik, J., and Horváth, I. (2019). From Validation of Medical Devices towards Validation of Adaptive Cyber-Physical Systems, Journal of Integrated Design & Process Science, Vol. 23, No. 1, pp. 37-59.

[37] Filieri, A., Tamburrelli, G., & Ghezzi, C. (2016). Supporting self-adaptation via quantitative verification and sensitivity analysis at run time. IEEE Transactions on Software Engineering, Vol. 42, No. 1, pp. 75-99.

[38] Muccini, H., Sharaf, M., & Weyns, D. (2016). Self-adaptation for cyber-physical systems: A systematic literature review, In: Proceedings of SEAMS'16, Austin, TX, pp. 75-81.

[39] Bianculli, D., Filieri, A., Ghezzi, C., & Mandrioli, D. (2013). A syntactic-semantic approach to incremental verification. arXiv preprint arXiv:1304.8034, pp. 1-22.

[40] Ghezzi C., Sharifloo A.M., Menghi C. (2013). Towards agile verification. In: Perspectives on the Future of Software Engineering. Springer, Berlin, Heidelberg, pp. 31-47.

[41] Jiang, Y., Song, H., Wang, R., Gu, M., Sun, J., & Sha, L. (2017). Data-centered runtime verification of wireless medical cyber-physical system. IEEE Transactions on Industrial Informatics, Vol. 13, No. 4, pp. 1900-1909.

[42] Pinisetty, S., Jéron, T., Tripakis, S., Falcone, Y., Marchand, H., & Preoteasa, V. (2017). Predictive runtime verification of timed properties. Journal of Systems and Software, Vol. 132, pp. 353-365.

[43] Bauer, A., Leucker, M. & Schallhart, C. (2011). Runtime verification for LTL and TLTL, ACM Transactions on Software Engineering and Methodology, Vol. 20, No. 4, p. 14, pp. 1-64.

[44] Calinescu, R., Ghezzi, C., Kwiatkowska, M., & Mirandola, R. (2012). Self-adaptive software needs quantitative verification at runtime. Communications of the ACM, Vol. 55, No. 9, pp. 69-77.

[45] Mitsch, S., & Platzer, A. (2016). ModelPlex: Verified runtime validation of verified cyber-physical system models. Formal Methods in System Design, Vol. 49, No. 1-2, pp. 33-74.