Meta-Adaptation Strategies for Adaptation in Cyber-Physical Systems

Ilias Gerostathopoulos, Tomas Bures, Petr Hnetynka, Adam Hujecek, Frantisek Plasil, and Dominik Skoda

Charles University in Prague, Faculty of Mathematics and Physics, Czech Republic

{iliasg, bures, hnetynka, plasil, skoda} @d3s.mff.cuni.cz

Abstract. Modern Cyber-Physical Systems (CPS) not only need to be dependable, but also resilient to and able to adapt to changing situations in their environment. When developing such CPS, however, it is often impossible to anticipate all potential situations upfront and provide corresponding tactics. Situations that lie out of this "envelope of adaptability" can lead to problems that range from single component malfunctioning to complete system failure. The existing approaches to self-adaptation cannot typically cope with such situations as they still rely on a fixed set of tactics, which in case of complex systems does not guarantee achieving correct functionality. To alleviate this problem, we propose the concept of meta-adaptation strategies, which extends the limits of adaptability of a system by constructing new tactics at runtime to reflect the changes in the environment. The approach is demonstrated on an existing architecture-based self-adaptation method and exemplified by two concrete meta-adaptation strategies.

Keywords: meta-adaptation strategies; adaptation tactics; cyber-physical systems

1 Introduction

An important feature of efficient and dependable CPS is self-adaptivity, i.e., the ability to change their behavior or structure in response to changes in their environment. Self-adaptation in software systems is usually achieved in three fundamental ways: (i) by relying on a detailed application model, e.g., Markov Decision Processes (MDP), and employing simulations or other means of state-space traversal to infer the best response of the system, (ii) by identifying control parameters and employing feedback-based control techniques from control theory, and (iii) by reconfiguring architecture models, typically with the help of Event-Condition-Action rules – *architecture-based self-adaptation*.

Existing approaches. A common denominator for all these three fundamental ways is that they monitor the state of the environment and select an operation to perform from a pre-designed fixed set of actions. In (i), a model of the environment is assumed to be available (either known or learned) and the self-adaptation selects an action (e.g., "go straight", "turn left", "turn right") to maximize future reward. In (ii) a fixed set of con-

trol parameters is given and the actions consist of setting (increasing/decreasing) a parameter value (e.g., Java heap size). In (iii), self-adaptation rules are expressed as actions involving particular architecture reconfigurations applicable under certain conditions in the presence of certain events or situations [1, 2]. The combination of Rainbow framework with the Stitch language is representative of (iii). In Stitch, a *tactic* is a specification of an activity with a pre- and post-condition and an associated action. The self-adaptation in (iii) can be thus seen as selecting one or more tactics from a fixed set.

These three ways have also been used both combined and together with learning-based approaches. For example, control theory has been employed in the runtime modification of the probabilities of a MDP [3]. Learning-based approaches have been proposed to deduce the impact of adaptation actions at runtime [4], and to mine the application model from system execution traces [5].

In the realm of CPS, where we deal with large complex distributed systems, the high level view of architecture-based self-adaptation (i.e., (iii)) is generally favored [1, 2, 6, 7]. At the same time, due to external uncertainty [8] (e.g., hardware failures, temporary network unavailability), anticipating all potential situations upfront is not an option. As a result, adapting by switching between available tactics applicable in different situations is problematic, as the CPS may arrive in a situation where no combination of tactics applies. A similar problem of selecting only from fixed actions and parameters applies also to (i) and (ii).

Goals. As a remedy, focusing specifically on architecture-based self-adaptation, we propose to generate new tactics at runtime to reflect the changes in the environment and increase the overall system utilities, in particular safety, performance, and availability. We do so by introducing the concept of meta-adaptation strategies (MAS). MAS allow us to enrich the adaptation logic of the system (thus the "meta" prefix) by systematically generating new tactics. This provides a dynamic space of actions and effectively extends the limits of adaptability of the system.

In particular, we present the idea of MAS and define their structure similar to design and adaptation patterns. On top of this basis, we show two examples of MAS and demonstrate their applicability. The two MAS examples of course do not cover the whole space of potential MAS, however, we believe that by introducing the idea of MAS as means for dynamically extending the limits of systems adaptability, we provide helpful inspiration for future research on self-adaptive systems.

2 Running Example and Background

To demonstrate the concept of MAS, we briefly overview below the running example and the IRM-SA self-adaptation method along with the DEECo component model, which serve as the model and technological basis we use to exemplify MAS.

Running Example: Firefighter Coordination System. Firefighters belonging to tactical groups are deployed on the emergency field and communicate via low-power nodes integrated into their personal protective equipment. Each of these nodes is configured

at runtime depending on the task assigned to its bearer. For example, a hazardous situation might need closer monitoring of a certain parameter (e.g., temperature).

In the setting of the complete case study [9], firefighters have to communicate with the officers (their group leaders), who are equipped with tablets; the software running on these tablets provides a model of the current situation (e.g., on a map) based on data measured at and aggregated from the low-power nodes. Parameters measured at each low-power node are *position*, external *temperature*, *battery level*, and *oxygen level*. The data aggregation on the side of the group leaders is done with the intention that each leader can infer whether any of his/her group members is in danger and take strategic decisions. Such a coordination system has increased safety and performance requirements. It needs to operate on top of opportunistic ad-hoc networks, where no guarantees for end-to-end response time exist, with minimum energy consumption, and without jeopardizing its end-users. It also needs to respond to a number of challenging situations: What if the temperature sensor starts malfunctioning or completely fails at runtime? What if firefighters are deployed inside a building where GPS readings are not available? What if the communication between members and their leader is lost?

In all these situations, each node has to adapt its behavior according to the latest information available. For example, if a firefighter node detects that it is in the situation "indoors", it has to switch from the tactic of determining the position via the GPS to the tactic of using an indoors tracking system. Other tactics include increasing the sensing rate in face of a danger or even relying on the nearby nodes for strategic actions when communication with the group leader is lost.

Obtaining an *exhaustive* list of situations that trigger adaptations in the firefighter coordination system is not a realistic option, as the environment is highly dynamic and unpredictable. We rather need to be able to build a system that would dynamically change its behavior by (i) generating new tactics on demand, and (ii) using them in the adaptation actions in order to deal with unanticipated situations.

IRM-SA and DEECo. The Invariant Refinement Method for Self-Adaptivity (IRM-SA) [9, 10] is a requirements-oriented design method tailored for the CPS domain. IRM-SA captures goals and requirements of the systems as *invariants* that describe the desired state of the system-to-be at every time instant. For example, consider invariant (1) in Fig. 1, which specifies that the leader of each firefighter group should have an up-to-date view (encapsulated in the positionMap field) of his/her group members. This "necessity" is AND-decomposed into invariants (2) and (3), which specify the necessities of propagating the position from each member to the leader and determining the position on the side of each member, respectively. The refinement is finished when each leaf invariant of the refinement tree is either an *assumption* or is a computation activity corresponding to a *process* or *knowledge exchange*. Alternative designs are captured by the OR-decomposition pattern, where each variant is guarded by an assumption capturing the state of the environment. For example, invariant (3) can be satisfied either by determining the position through an indoors tracking system – invariant (5) – or a global positioning system – invariant (7). At runtime, the system monitors the satisfaction of assumptions (4) and (6) and activates the activity corresponding to the chosen branch in the tree.

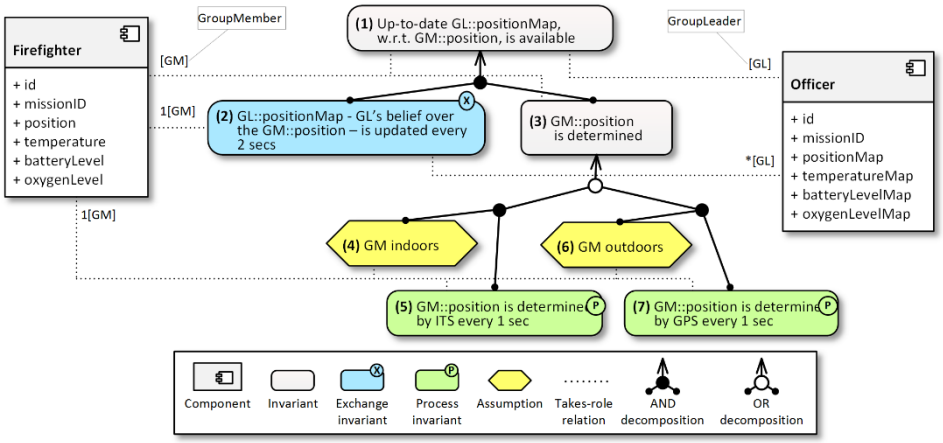

Fig. 1. Excerpt from the IRM-SA model of the running example.

Although IRM-SA is a method that can be used independently, it is very well aligned with the DEECo component model [11]. DEECo features autonomous components forming dynamic goal-driven collaboration groups – *ensembles*. Components contain knowledge (their data) and processes, whose periodic execution results in periodic updates in their knowledge. Components are not bound to each other; they can only indirectly communicate within an ensemble. The communication takes the form of mapping a component's knowledge field into another component's knowledge field – *knowledge exchange*. Membership of a component in an ensemble is not static, but periodically evaluated at runtime based on a condition specified over ensemble-specific interfaces that provide partial views over the components' knowledge.

In the IRM-SA–to–DEECo mapping, IRM-SA components correspond to DEECo components; process invariants to component processes; exchange invariants to ensembles; and assumptions to DEECo runtime monitors. The IRM-SA graph corresponds to the adaptation logic that DEECo applications use in order to switch on and off certain features (according to the branches selected at runtime on the IRM-SA graph). In this frame, an adaptation action is choosing an applicable configuration by choosing among the branches in the IRM-graph, whereas a tactic corresponds to an individual leaf invariant. An adaptation action thus consists of selecting a set of tactics.

3 Meta-Adaptation Strategies

As already discussed in the previous sections, a system, even a self-adaptive one, can be designed to handle a limited number of runtime situations. Interestingly, the number of new distinct tactics that can be devised in response to an unanticipated situation is in principle infinite. Hence, apart from being able to devise new tactics, it is important to be able to rank them according to their effect on the system, in order to be able to select the most promising one, or, at least, to select the ones that are worth trying.

(Here, we assume the general adaptation loop in which the adaptation mechanism activates a tactic, observes its effect on the system and, depending on its impact, either keeps it or tries another tactic.)

To systematize the creation of new tactics, we rely on the concept of meta-adaptation strategies (MAS). MAS serve as patterns for extending the limits of adaptability of the system, with each strategy extending the limits in a certain way. The goal of such a strategy is twofold:

1. *To provide an algorithm to systematically generate a set of tactics at runtime.*
2. *To provide a metric on the generated set according to which the tactics can be ranked.*

In the rest of the section, we exemplify our proposal by describing two MAS. Note that the two strategies can be applied sequentially or in parallel in the running system, since they are by design orthogonal to each other. Due to space constraints, we omit a third strategy that we have so far developed – details on that can be found in [12].

Tactics Generated by Data Classification. In CPS, exploiting the interdependencies between sensed data is an opportunity for introducing specific meta-adaptation strategies. A particular case is the location-dependency of data, i.e., the fact that the value of certain measurable system attributes depends on the physical location of the sensors that provide the data. Below we describe a strategy providing a way to automatically create knowledge exchange specifications (*ensemble specifications* in DEECo) that introduce "collaborative sensing" (when direct sensing is not possible anymore) and feed them into the running system. Hence, such new ensembles represent new tactics.

Strategy Name: Knowledge Exchange by Data Classification

Intent: To increase the robustness of the system, prolong its acceptable functioning, or achieve graceful degradation in face of data unavailability and outdatedness.

Context: The strategy targets the case when values of a knowledge field of a component become outdated to the extent that they cannot be relied upon in terms of correct behavior of the component. For instance, there is a sensor malfunction that prevents value updates.

Behavior: To make up for losing the ability to obtain the actual value of an outdated knowledge field, create a new ensemble specification through which the field is assigned an approximated value based on the up-to-date related knowledge values of other components. This specification consists of (i) a membership condition, which prescribes the condition under the components should interact, and (ii) a knowledge exchange function, which specifies the knowledge exchange that takes place between the collaborating components. (For simplicity, we consider the knowledge exchange that just copies the data without manipulating them in any other way.)

To be able to construct the membership condition when the situation targeted by the strategy happens, observe first the system when it is healthy and log components' knowledge (as a time series of the knowledge evolution). Analyze (typically offline) the logged knowledge and find conditional correlations indicating that when values of some knowledge fields $A_1, B_1, \dots, A_n, B_n$ are pairwise "close" then other values of other knowledge fields C, D are "close" as well. Formulate an ensemble (to be instantiated

when the situation targeted by the strategy happened), which uses the pairwise "closeness" of $A_1, B_1, \ldots, A_n, B_n$ as the membership condition and has the assignment $D := C$ as the knowledge exchange.

Generate a number of possible membership conditions corresponding to different tactics. Then, select a tactic by applying the metric of selecting the tactic which provides the most general condition given the target confidence level.

Contraindications: The analysis of the collected time series can be very resource demanding and therefore a dedicated hardware infrastructure should be used. Similarly, the data collection may be a rather resource-intensive process, especially when components' knowledge is big or changes frequently. Also, introducing superfluous new ensembles can overload the system with unnecessary replicated data.

Example: In the firefighter coordination case study, each firefighter component features the knowledge fields of position and temperature. Suppose that the temperature values are used to control the suit cooling system. Obviously, when the temperature sensor breaks, a real-life threat arises. Since firefighters are usually moving in groups so that those close to each other obtain similar temperature readings, the temperature value of one component can be approximated based on the temperature values of the others, when their positions are close. Technically, the threshold of temperature proximity can be set (e.g., 20°C).

Tactics Generated by Period Adjusting. A CPS typically brings real-time requirements that are reflected in schedulability parameters of component processes. The schedulability parameters can be typically inferred by real-time design via schedulability analysis. However, when schedulability parameters influence the systems in a complex manner (e.g., when there is a tradeoff between CPU utilization, battery, network utilization), it is not possible to infer them by systematic analysis. Rather, the schedulability parameters are set manually, based on the experience of the system's architect. The strategy below addresses the case when the manually set schedulability parameters cannot cope with an unanticipated situation.

Strategy Name: Process Period Adjusting

Intent: To optimize the scheduling of processes with respect to overall system (application-specific) performance in a system where processes are scheduled periodically.

Context: The strategy targets situations when the system starts failing due to violated timing requirements and the schedulability parameters cannot be inferred a priori because they influence the system in a complex manner.

Behavior: Let R be the set of all active real-time processes in the system. To be able to identify the situation when a requirement for a process r_i in R with period p_i is not satisfied anymore, equip each r_i with a runtime monitor returning a fitness value f_i (real number in [0-1]). Generate tactics that correspond to a new real-time processes r_i' created from r_i by adjusting (reducing or enlarging within pre-defined permissible bounds) p_i to p_i', when f_i drops below an acceptable threshold. To explore the search space of possible period adjustments, employ the genetic algorithm (1+1)-ONLINE EA [13]. Changing p_i can be interpreted as generating a new tactic r_i' and using it to substitute the tactic r_i in the system. Terminate the period adjusting procedure when the adjustment of each p_i has been exercised in both directions and there is no further benefit.

In this strategy, tactics (new processes) are compared by substituting them to the running system and calculating the overall system fitness as a function of f_i's.

Contraindications: Reducing periods (a usual action) may have a negative impact on other resources (CPU, battery, network). In such a case, the impact would have to be modelled and taken into consideration in the state-space search.

Example: Consider extending the design of our running example by a root invariant that specifies that "battery consumption should be kept minimized". In order to satisfy this invariant, the system will try at runtime to tweak the processes' periods to invoke them as scarcely as possible. At the same time, when there is high inaccuracy in the GPS readings (e.g., less than 3 satellites in sight), the GPS process may need to be invoked more often to make sure the cumulative inaccuracy of the estimated position of a moving firefighter is within certain bounds. (The cumulative inaccuracy is essentially the sum of the initial inaccuracy of the GPS reading and the distance a firefighter has moved since the last GPS reading.) It is thus a dynamic trade-off between availability and dependability that has to be resolved at runtime.

4 Experimental Evaluation and Conclusion

In order to evaluate the feasibility of the proposed MAS, we implemented them as extensions of the IRM-SA jDEECo plugin[1]. Our evaluation scenario consisted of three firefighters moving in a building, periodically monitoring their battery level, position, and external temperature. The objective of the system was to obtain accurate enough values of position and temperature, while keeping battery consumption minimal. Two malfunctions were introduced: (i) the GPS of one of the firefighters became inaccurate, and (ii) the temperature sensor of the firefighter was broken completely.

The MAS described in the paper were able to successfully cope these unanticipated malfunctions – the "Process Period Adjusting" reduced the inaccuracy stemming from knowledge outdatedness thus compensating the inaccuracy of the GPS reading; the "Knowledge Exchange by Data Classification" created and deployed a new ensemble, which provided a temperature estimation to compensate for the broken sensor. The evaluation, together with all the measurements, is described in detail in [12].

Conclusion. In this paper, we suggested a way to address the problem of limited adaptability caused by a fixed set of tactics. To this end, we have introduced the concept of meta-adaptation strategies (MAS) as a means for creating new tactics by observing the behavior of a system at runtime. In addition to laying out the general concept of MAS, we have exemplified the concept by two instances of MAS. Generally, if a system is subject to environment uncertainty, the extent of the problem space that should be covered by systems adaptability is unknown. This makes it impossible to devise all adaptation tactics at design time. It of course makes it also impossible to presume all necessary meta-adaptation strategies, as each strategy covers only a certain sub-space of the problem space. However, compared to pre-designed tactics, the meta-adaptation strat-

[1] https://github.com/d3scomp/IRM-SA/tree/ECSA2015

egy involves observation of system's and environment's evolution at runtime and utilizes this to formulate new tactics. As such, it has the potential to carry through higher expressive power than pre-designed tactics and consequently achieve significantly higher coverage of the problem space.

Acknowledgements. The work on this paper has been supported by Charles University institutional funding SVV-2015-260222.

References

1. Cheng, S.-W., Garlan, D., Schmerl, B.: Stitch: A language for architecture-based self-adaptation. J. Syst. Softw. 85, 1–38 (2012).
2. David, P.-C., Ledoux, T., Léger, M., Coupaye, T.: FPath and FScript: Language support for navigation and reliable reconfiguration of Fractal architectures. Ann. Telecommun. 64, 45–63 (2009).
3. Filieri, A., Ghezzi, C., Leva, A., Maggio, M., Milano, P.: Self-Adaptive Software Meets Control Theory: A Preliminary Approach Supporting Reliability Requirements. In: Proc. of ASE '11. pp. 283–292. IEEE (2011).
4. Elkhodary, A., Esfahani, N., Malek, S.: FUSION: A Framework for Engineering Self-tuning Self-adaptive Software Systems. In: Proc. of FSE '10. pp. 7–16. ACM (2010).
5. Yuan, E., Esfahani, N., Malek, S.: Automated Mining of Software Component Interactions for Self-Adaptation. In: Proc. of SEAMS '14. pp. 27–36. ACM (2014).
6. Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B., Steenkiste, P.: Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. Computer. 37, 46–54 (2004).
7. Hirsch, D., Kramer, J., Magee, J., Uchitel, S.: Modes for Software Architectures. In: Proceedings of EWSA'06. pp. 113–126. Springer (2006).
8. Esfahani, N., Kouroshfar, E., Malek, S.: Taming uncertainty in self-adaptive software. In: Proc. of SIGSOFT/FSE '11. pp. 234–244. ACM (2011).
9. Gerostathopoulos, I., Bures, T., Hnetynka, P., Keznikl, J., Kit, M., Plasil, F., Plouzeau, N.: Self-Adaptation in Cyber-Physical Systems: from System Goals to Architecture Configurations. Department of Distributed and Dependable Systems, D3S-TR-2015-02 (2015).
10. Keznikl, J., Bures, T., Plasil, F., Gerostathopoulos, I., Hnetynka, P., Hoch, N.: Design of Ensemble-Based Component Systems by Invariant Refinement. In: Proc. of CBSE'13. pp. 91–100. ACM (2013).
11. Bures, T., Gerostathopoulos, I., Hnetynka, P., Keznikl, J., Kit, M., Plasil, F.: DEECo – an Ensemble-Based Component System. In: Proc. of CBSE'13. pp. 81–90. ACM (2013).
12. Gerostathopoulos, I., Bures, T., Hnetynka, P., Hujecek, A., Plasil, F., Skoda, D.: Meta-Adaptation Strategies for Adaptation in Cyber-Physical Systems. Department of Distributed and Dependable Systems, D3S-TR-2015-01 (2015).
13. Bredeche, N., Haasdijk, E., Eiben, A.E.: On-Line, On-Board Evolution of Robot Controllers. In: Collet, P., Monmarché, N., Legrand, P., Schoenauer, M., and Lutton, E. (eds.) Artifical Evolution. pp. 110–121. Springer Berlin Heidelberg (2010).