Using Learning Classifier Systems for the DSE of Adaptive Embedded Systems

Fedor Smirnov, Behnaz Pourmohseni, Jürgen Teich
Hardware/Software Co-Design, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany
{fedor.smirnov, behnaz.pourmohseni, juergen.teich}@fau.de

Abstract*—Modern embedded systems are not only becoming more and more complex but are also often exposed to dynamically changing run-time conditions such as resource availability or processing power requirements. This trend has led to the emergence of adaptive systems which are designed using novel approaches that combine a static off-line Design Space Exploration (DSE) with the consideration of the dynamic run-time behavior of the system under design. In contrast to a static design approach, which provides a single design solution as a compromise between the possible run-time situations, the off-line DSE of these so-called *hybrid* design approaches yields a set of configuration alternatives, so that at run time, it becomes possible to dynamically choose the option most suited for the current situation. However, most of these approaches still use optimizers which were primarily developed for static design. Consequently, modeling complex dynamic environments or run-time requirements is either not possible or comes at the cost of a significant computation overhead or results of poor quality. As a remedy, this paper introduces *Learning Optimizer Constrained by ALtering conditions (LOCAL)*, a novel optimization framework for the DSE of adaptive embedded systems. Following the structure of *Learning Classifier System (LCS)* optimizers, the proposed framework optimizes a *strategy*, i.e., a set of conditionally applicable solutions for the problem at hand, instead of a set of independent solutions. We show how the proposed framework—which can be used for the optimization of any adaptive system—is used for the optimization of dynamically reconfigurable many-core systems and provide experimental evidence that the hereby obtained strategy offers superior embeddability compared to the solutions provided by a s.o.t.a. hybrid approach which uses an evolutionary algorithm.

***Index Terms*—DSE, Adaptive Systems, Embedded Systems, Learning Classifier Systems.**

I. Introduction and Related Work

Recent advances in process technology downsizing enable the integration of an extensive number of processing elements on a single chip, giving rise to embedded systems with an unprecedented processing power. Alongside the development of novel applications with high computational requirements, the flexibility resulting from this newly available processing power is used for the development of *adaptive* systems. In contrast to *static* systems, which are developed for one single application and have a limited interaction with their environment, adaptive systems can react to different Operational Conditions (OCs) by a dynamic adaptation, e.g., a reconfiguration or a switch to a different operational mode. Hereby, a so-called *adaptation manager* decides how the system adapts to a given OC.

The design of the adaptation manager—which is typically represented as a *dynamic optimization problem*—is one of the most important steps during the design of an adaptive system, since the choice of the appropriate action for a given OC directly determines how well the adaptive system can react to run-time events while still providing a high quality of service and satisfying non-functional requirements such as real-time constraints. The scientific community has, therefore, invested great research effort in dynamic optimization problems. In recent years, new approaches have been developed for dynamic distance minimization [6], [7], optimization of constrained dynamic problems [2], [14], and the run-time optimization of operating systems [13], [15]. These approaches, however, share the assumption that the optimization takes place during the operation of the optimized system, i.e., at run time, which restricts their applicability to systems that (a) can offer the computational power required for the optimization process and (b) are not subject to strict real-time requirements, since every adaption (in response to changing OCs) involves a time-consuming optimization to find the new system configuration.

In the embedded domain, the inherent system dynamism has, for a long time, been addressed by similar approaches which optimize the system configuration at run time [1], [3], [19]. However, due to the processing power restrictions of embedded systems, these so-called *dynamic* approaches cannot make use of the aforementioned computationally intensive on-line optimization. Instead, they have to resort to lightweight optimization heuristics, which, in turn, degrades their (a) responsiveness to environmental changes and (b) ability to find high-quality system configurations. On the other hand, so-called *static* design approaches—which have become the de facto standard practice for the design of embedded systems with restricted dynamism—overcome the problem of limited processing power by making all design decisions within a computationally intensive off-line Design Space Exploration (DSE) [18], [23], [30]. However, these approaches generate a single system configuration, which offers a tolerable compromise for the expected OCs, and, consequently, cannot address highly dynamic environments.

The recently emerged *hybrid* approaches address the design of adaptive embedded systems by combining the advantages of static and dynamic design approaches[1]. These approaches use an off-line DSE to generate a set of configuration options which address various run-time scenarios. At run time, the adaptation manager—which, in the context of adaptive embedded systems, is typically referred to as the *Run-Time Manager (RTM)*—performs a lightweight check of the OCs and selects one of the precalculated configurations that matches the current situation. Hybrid design approaches enable coping with the high number of possible run-time conditions and have been successfully used for the design of adaptive embedded systems. However, while the system models of

[1]An extensive overview on existing static, dynamic, and hybrid embedded system design approaches is provided in [20].

these approaches are able to capture the inherent dynamism of adaptive systems, the majority of hybrid approaches rely on the same optimization algorithms as static design approaches, e.g., Evolutionary Algorithms (EAs), Simulated Annealing (SA), or Particle Swarm Optimization (PSO). These optimizers, which are referred to as Classic Optimizers (COs) in this paper, have been the focus of extensive research for many years, resulting in an excellent performance for static design problems. However, despite their capability to provide multiple solutions in case of a multi-objective problem, COs optimize each configuration generated during the DSE individually and in complete isolation from the other configurations. Consequently, modeling the fact that the generated configurations are to be used as a set of conditionally applicable alternatives is either not possible or requires in-depth knowledge of the optimization problem and comes at the cost of a significant performance loss, e.g., in cases where the dimensionality of the optimizer's objective space is increased by additional objectives reflecting the run-time situation. Moreover, COs have an inflexible structure and require unambiguous functions to evaluate the configuration quality, which makes it difficult to reflect complex run-time situations, abstract design goals, and dynamic conditions in their optimization model. Due to the limited adaptability of their optimization models and their specialization on static problems, COs, thus, cannot be trivially adopted for dynamic optimization problems.

Contribution: As a remedy, we propose *Learning Optimizer Constrained by ALtering conditions (LOCAL)*, a novel optimization framework for the DSE of adaptive embedded systems. In contrast to s.o.t.a. approaches for the design of adaptive systems, LOCAL does not rely on computationally intensive learning processes at run time. Instead, the proposed framework transfers the learning process of the adaptive system from the run time to the design phase by emulating the interaction between the environment and the adaptation manager during the off-line design. Hereby, the configurations addressing a given OC are generated within a *constrained DSE*, which is restricted to search only the space of configurations that are applicable to the given OC. LOCAL considers both external events and the way that the adaptive system influences its surroundings. In its structure, LOCAL is inspired by Learning Classifier Systems (LCSs), optimization algorithms developed specifically for complex and dynamically changing problems [8], [22], [24], [26], [27], [29]. Instead of a set of individual system configurations, LOCAL provides a rule-based adaptation strategy—which defines the system's run-time behavior and can be implemented using a lightweight RTM—as the optimization result.

In this paper, the structure and the components of the LOCAL framework—which can be used for the optimization of any dynamic problem—are illustrated by applying it to the optimization of an adaptive many-core system with a dynamic deployment of applications. While the reconfigurability of these systems offers many practical advantages, their high degree of dynamism makes their design very challenging. The implementation of this problem within LOCAL highlights the expressiveness of the proposed optimization model as a major advantage of LOCAL compared to COs. In the experiment section of this paper, the application deployment strategy generated by LOCAL is compared to the set of deployment options provided by a s.o.t.a. hybrid design approach relying on an EA. The experimental results give evidence that modeling the dynamic problem within LOCAL enables the optimizer to provide a strategy which is superior w.r.t. embeddability, an abstract but crucial design objective which is difficult to describe by an evaluation function.

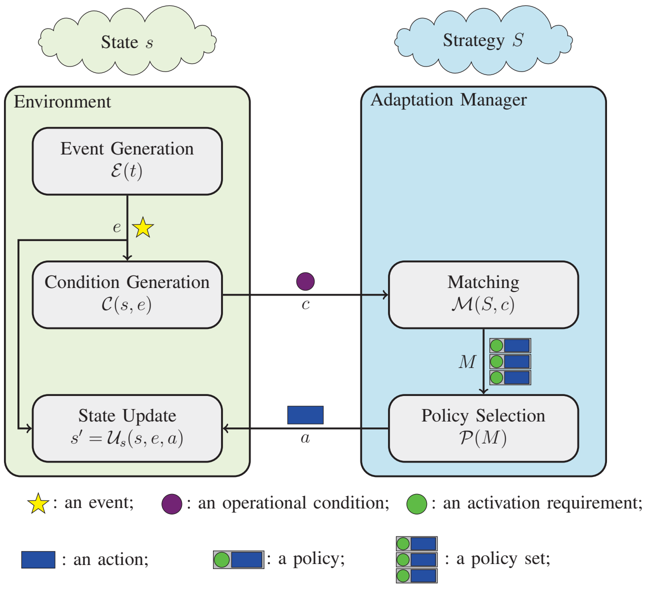
State s
Strategy S
Environment
Adaptation Manager
Event Generation $\mathcal{E}(t)$
e
Condition Generation $\mathcal{C}(s,e)$
c
Matching $\mathcal{M}(S,c)$
M
State Update $s' = \mathcal{U}_s(s,e,a)$
a
Policy Selection $\mathcal{P}(M)$
: an event;
: an operational condition;
: an activation requirement;
: an action;
: a policy;
: a policy set;

Fig. 1. Illustration of the data flow within one interaction between the environment and the adaptation manager.

II. The LOCAL framework

A. Model of the Adaptive System

In this paper, the behavior of an adaptive system is modeled as an interaction between (a) an *environment* which provides the *OCs* for the system and (b) an *adaptation manager* which generates the appropriate *action* taken by the system, see Fig. 1. The OCs encompass any information which is relevant for the management of the system such as user input, application workload, current allocation of resources, or the failure of individual components. The actions model the steps that are taken to adapt the system to a given OC, e.g., a reconfiguration or an adjustment of system parameters.

1) The Environment: At any given point in time, the environment provides an OC c, which is generated by the function $\mathcal{C}(s,e)$ based on the current environment state s and the current external event(s) e. The external events e are generated by the function $\mathcal{E}(t)$ and capture any system inputs which are relevant for the OCs, but do not depend on the current state of the system. Examples of external events include user input or a varying workload of an application. The state s, on the other hand, models any characteristics of the system itself which (a) affect the OCs and (b) can be altered consequent to external events e and the action a taken by the adaptation manager. Every interaction between the environment and the adaptation manager ends with the update of the system state, where the next state s' is determined by the function $\mathcal{U}_s(s,e,a)$.

2) The Adaptation Manager: At run time, the adaptation manager reacts to the given OC c by taking an action a, following the *strategy* S. The strategy is given as a set of *policies*. Hereby, each policy p is an if-then rule, represented by the tuple (r, a), where r is the activation requirement and a is the corresponding action. An activation requirement describes the preconditions for the execution of the corresponding action. The function $\mathcal{A}(r, c)$ captures the relation between an activation requirement r and an OC c and evaluates to 1 iff r is fulfilled in situations described by c.

After receiving the current OC c, the adaptation manager generates the so-called *match set* M by applying the function $\mathcal{M}(S, c)$. During this step, the function $\mathcal{A}(r, c)$ is applied to the activation requirement of each policy $p \in S$. All policies whose activation requirement if fulfilled by c are added to the match set: $\forall p = (r, a) \in S : p \in M \iff \mathcal{A}(r, c) = 1$. The match set M, therefore, contains all policies which are part of the current strategy and are applicable under the given OC.

After the matching step, the adaptation manager chooses the action to be taken by applying the selection function $\mathcal{P}(M)$ on the match set M. The selected action a is then provided to the environment, triggering an update of the system state.

B. Design-Time Optimization

In this paper, we, w.l.o.g., assume that the adaptation manager is equipped with a fix strategy, which is not altered at run time. At design time, the proposed framework, LOCAL, is used to generate a strategy which is tailored to the expected run-time conditions. To this end, the information available at design time is used to model the expected environment within the framework. By emulating the iterative run-time interaction between the environment and the adaptation manager, LOCAL generates a strategy which considers the expected external events and the way in which the chosen actions and the external events affect the system state.

1) Optimization Flow: Figure 2 illustrates the components used during the design-time optimization of the strategy. In addition to the components modeling the run-time behavior of the system (gray blocks in Fig. 2) LOCAL features components (orange blocks) which are used to optimize policies for the current OC. Just like in the model of the run-time system detailed in the previous section, each interaction between the environment and the adaptation manager starts with the generation of external events e, followed by the generation of the OC c. After the matching step, where the current strategy is scanned for policies matching c, LOCAL—according to an explore/exploit ratio specified by the designer—decides whether in the current iteration (a) only policies which are part of the hitherto generated strategy are considered for the policy selection (exploit) or (b) a set of new policies, M_x, is generated for the provided OC (explore). For the exploration of new policies—which is always executed in cases where the current strategy does not contain any matching policies for the provided condition—LOCAL uses a *constrained DSE*, denoted as function $\mathcal{D}(M, c)$. The DSE explores a search space restricted to matching actions in quest of actions which are Pareto-optimal w.r.t. a set of design objectives. The new policies comprised in the set M_x are used to update the strategy—where they possibly replace other policies—and are considered during the policy selection step.

2) Constrained DSE: The constrained DSE used by LOCAL is based on SAT Decoding [10], [21], an approach for the optimization of constrained combinatorial problems. In SAT Decoding, a set of Pseudo-Boolean SAT constraints restricts the search space of the optimizer, so that only problem solutions which satisfy the constraint set are considered during the DSE. When applied for the design of static embedded systems, SAT Decoding is typically used with a constraint set which encodes a valid task mapping, a valid resource allocation, and a valid message routing, hereby restricting the DSE to the so-called *solution space* $\mathfrak{S}$. In addition to these so-called *problem constraints*, LOCAL encodes *condition constraints* which restrict the search space to the *match space* $\mathfrak{M}$, so that all problem solutions which are not applicable w.r.t. the given OC are excluded from the exploration. Any solution found during the exploration of the *action space* $\mathfrak{A}$—defined by the union of problem and condition constraints—is valid and can be used to create a policy matching the current OC.

III. Practical Example

To provide a concrete demonstration of LOCAL's functionality, this section details how it can be used for the design-time optimization of an adaptive many-core system with dynamic application deployment. After describing the motivation and the goals of this dynamic optimization problem, we demonstrate how it is implemented within LOCAL.

A. Problem Description

We use the off-line optimization of a heterogeneous many-core system with a dynamic deployment of hard real-time applications as an example for a dynamic optimization problem. In the considered system, a RTM is used to launch/terminate the applications on demand. To launch an application, the RTM must find a feasible *deployment* of the application in question, i.e., an assignment of the application's tasks to platform resources which (a) are not in use[2] by other applications running on the platform, (b) offer enough computational power (for the timely execution of tasks), and (c) are in a sufficiently close physical proximity to each other (for short transmission times of the messages) to satisfy the hard deadline of the application. Since performing a timing analysis at run time is not possible due to the RTM's limited computational power, it must be provided with a set of deployment options which are (a) generated in a computationally intensive off-line DSE and (b) guaranteed to satisfy the application's deadline. The **main goal** of the off-line DSE is to provide a set of deployment options with a **high *embeddability***. Hereby, the embeddability of a deployment option can be quantified by the likelihood that, by using this option during a run-time situation, the RTM can successfully deploy the application. While it is possible to provide the RTM with a set of concrete deployment options (binding the tasks to concrete cores on the platform), the authors of [25] have shown that providing the RTM with a set of so-called *Operating Points (OPs)* results in a significantly better embeddability. Instead of describing a concrete deployment option, an OP is given by a set of *deployment requirements*, namely the minimal number of required processor cores of each type, the so-called *clustering*

[2]We consider a *composable* application deployment, where isolation schemes are used to ensure that the deployment of an application instance does not affect the performance guarantees—e.g., real-time capability or energy consumption—of other applications which are embedded on the platform [16].

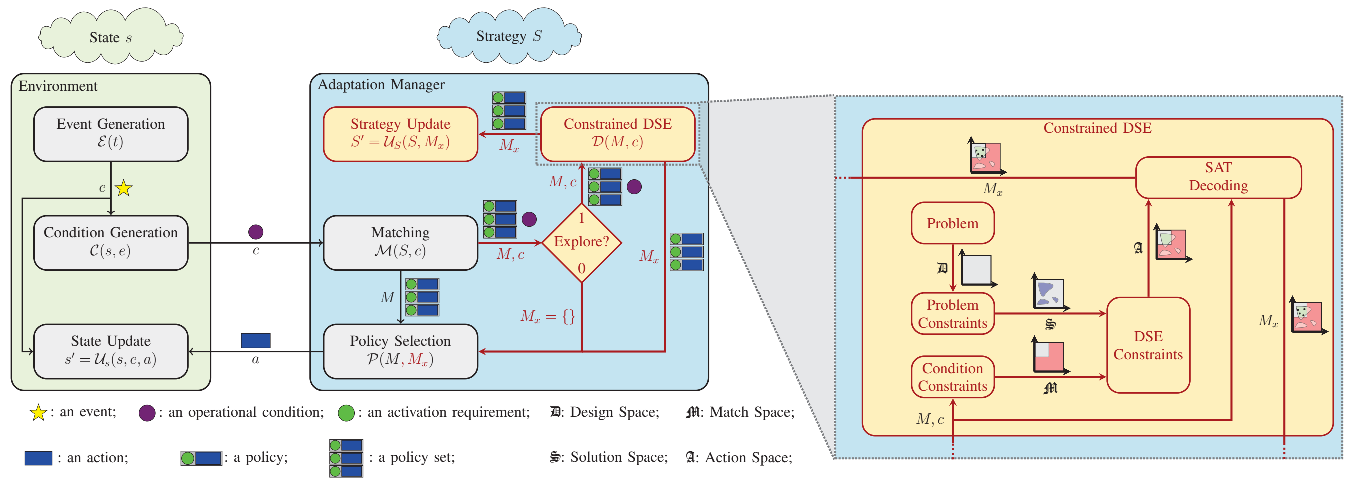
State s
Strategy S
Environment
Event Generation $\mathcal{E}(t)$
e
Condition Generation $\mathcal{C}(s,e)$
c
State Update $s' = \mathcal{U}_s(s,e,a)$
a
Adaptation Manager
Strategy Update $S' = \mathcal{U}_S(S, M_x)$
Constrained DSE $\mathcal{D}(M,c)$
M_x
M,c
Matching $\mathcal{M}(S,c)$
M
M,c
1
Explore?
0
M_x
$M_x = \{\}$
Policy Selection $\mathcal{P}(M, M_x)$
Constrained DSE
M_x
SAT Decoding
Problem
$\mathfrak{D}$
$\mathfrak{A}$
Problem Constraints
$\mathfrak{S}$
DSE Constraints
M_x
Condition Constraints
$\mathfrak{M}$
M,c
: an event; : an operational condition; : an activation requirement; $\mathfrak{D}$: Design Space; $\mathfrak{M}$: Match Space;
: an action; : a policy; : a policy set; $\mathfrak{S}$: Solution Space; $\mathfrak{A}$: Action Space;

Fig. 2. The data flow during one iteration of the design-time strategy optimization. The components which are present both at run and at design time are illustrated as gray blocks. The orange blocks illustrate the components which are only present at design time.

of tasks (each cluster denotes a set of tasks which must be bound onto the same core) and the maximal hop distance and minimal available communication bandwidth which is required for messages exchanged between task clusters [25]. Each OP corresponds to possibly many concrete deployments with guaranteed worst-case timing behavior, analyzed based on the deployment requirements. At run time, any concrete deployment which satisfies the requirements of an OP is guaranteed to satisfy the deadline of the application.

B. *Implementation within the LOCAL Framework*

Overall, the design-time optimization in LOCAL mimics a situation where multiple instances of different applications are concurrently deployed on the platform. Each event e generated by $\mathcal{E}(t)$ represents a user request for the deployment of one of the four applications from the E3S benchmarks suite [5]. The system state s captures the information about the resources and the links which are occupied by application instances deployed in previous optimization iterations. The OC c, therefore, corresponds to a deployment request for an application on a partially occupied platform. Each policy $p = (r, a) \in S$ is given by an OP of an application. The activation requirement r hereby corresponds to the deployment requirement of the OP, while the action a represents the actual deployment of the application instance. A policy p, therefore, matches a given OC iff it (a) addresses the requested application and (b) its OP can be deployed on a platform with an occupation given by the current state s. In cases where multiple policies in the strategy match c—i.e., the cases where the requested application can be deployed on the (partially occupied) platform in multiple ways—the policy selection $\mathcal{P}(M, M_x)$ chooses the policy whose OP has the lowest energy consumption. During the state update $\mathcal{U}_s(s, e, a)$, the platform occupation (state s) is updated by deploying the OP of the selected policy. For each policy in the strategy, a *usefulness score* is incremented every time that the policy is selected by $\mathcal{P}(M, M_x)$. In the context of the constrained DSE $\mathcal{D}(M, c)$ used for the generation of new policies, the problem constraints are a set of constraints which encode a valid *synthesis* of the requested application on the platform [12], while the condition constraints make sure that occupied resources and links cannot be used for the deployment of additional application instances. The constrained DSE, therefore, explores the space of valid deployments on the occupied platform in quest of a deployment option with a minimal energy consumption. During the strategy update $\mathcal{U}_S(S, M_x)$, the set of new policies M_x is added to the strategy, possibly replacing previously known policies with the lowest usefulness score. As soon as the platform occupation reaches a state where a deployment is not possible for any application, i.e., a state where no solution exists for the union of the problem and the condition constraints, the platform occupation is reset and the optimization continues with an unoccupied platform.

IV. Experimental Evaluation

For the evaluation of the proposed framework, we use an experiment where the run-time strategy of a RTM is optimized in order to provide a high embeddability (primary goal) and a low energy consumption (secondary goal). Before discussing the results, we provide details about the reference approach, the experiment setup, and the utilized tools.

A. *Reference Approach*

We use the *DAARM* approach [25] as a reference. In this hybrid approach, the OPs for the RTM are generated in an offline multi-objective DSE based on an EA. The authors quantify the quality of each OP by evaluating its makespan [16] and its energy consumption [28]. Furthermore, in order to obtain an OP set with a diverse set of deployment requirements, they define 6 additional objectives such as the required number of resources or the average hop distance of a message. Since an exploration with 8 objectives results in an extensive number of non-dominated solutions, a crowding-distance based truncation scheme is used to retain 100 solutions in the archive. The obtained solutions are then transformed into OPs by extracting their deployment requirements.

Compared to DAARM, LOCAL offers three advantages: (a) The expressiveness of its optimization model enables a direct modeling of the run-time situation, so that embeddability in different situations can be assessed without introducing 6 (!) additional objectives. (b) The emulation of the run-time situation enables generating a strategy tailored to deployment situations which are likely to happen with the given platform and the given set of applications. (c) Considering actual occupation scenarios enables a strong restriction of the DSE search space, resulting in solutions with a higher quality.

B. Experiment Setup

For the embeddability experiment, we consider a heterogeneous tiled many-core platform organized in 5×5 tiles interconnected by a NoC. We consider 4 cores per tile and 3 different core types. During the experiment, a simulated RTM uses the platform to deploy multiple instances of four applications from the E3S benchmarks suite [5]. Hereby, inter-application isolation is enforced by an exclusive allocation of cores per application instance [16]. For each of the four applications, the RTM is provided with a set of OPs generated by the evaluated approach. During each iteration of the experiment, the RTM randomly chooses one of the four benchmark applications and attempts to deploy it using its OPs. The order in which it attempts to deploy the OPs is hereby given by a list, where the OPs are sorted in an ascending order w.r.t. energy consumption. Consequently, the RTM always deploys the most energy-efficient OP that can be embedded in the current situation. Each successful deployment increases the platform occupation. As soon as no further instance of any application can be deployed, the platform occupation is reset and the experiment continues with an empty platform. Overall, we perform 1,000 deployment attempts for each of the two evaluated approaches, DAARM and LOCAL.

Both approaches provide the RTM with a set of OPs which dictates its adaptation strategy during the experiment. Hereby, the DAARM approach generates 100 OPs for each of the four applications in four separate optimization runs. The strategy provided by LOCAL, on the other hand, is generated within a single optimization run, considering that (a) the four applications are deployed in combination and (b) the RTM gives preference to energy-efficient OPs. LOCAL generates a strategy consisting of 80 OPs, as opposed to the 400 OPs provided by DAARM.

C. Implementation Details

The deployment of an OP is implemented as a resolution of a SAT constraint set given as a union of (a) the synthesis constraints from [12], (b) a set of constraints enforcing the satisfaction of the deployment requirements of the OP in question, and (c) a set of constraints reflecting the current platform occupation. A situation where the SAT4J solver [9] can find a solution is interpreted as a *successful* deployment, while a contradiction denotes a *failure*. In case of a successful resolution, the platform occupation is updated according to the deployment corresponding to the solution found by the solver. The off-line DSE of both approaches and, in particular, the enforcement of synthesis constraints (and, in case of LOCAL, condition constraints) is implemented using SAT-Decoding [10], [21], integrated in the OPENDSE [17] framework. An implementation of the EA NSGA-II [4] integrated in the OPT4J [11] framework is used as the main optimization mechanism of the DAARM approach and as the optimizer for the constrained DSE of LOCAL.

D. Result Discussion

The results of the embeddability experiment are illustrated in Fig. 3. The four columns show the embeddability for the four applications from the E3S benchmark suite, namely consumer, networking, telecommunication, and automotive. The scatter plots in the first and the second rows illustrate the deployment success and failure of the strategies generated by DAARM and by LOCAL, respectively, while the third row summarizes the individual deployment attempts to a success rate w.r.t. the level of resource utilization, i.e., the fraction of cores occupied at the time of the deployment attempt.

Both the scatter and the success rate plots clearly indicate that LOCAL provides a strategy with a superior embeddability. Apart from a single outlier (networking application at a resource utilization of around 65%), LOCAL's strategy offers a higher success rate of deployment attempts for all applications and all platform occupation levels. This is particularly true for the automotive application, where LOCAL's strategy offers a success rate advantage of up to 90%. Note that among the four benchmark applications, automotive consists of the largest number of tasks and also features the strictest deadlines. Deploying the automotive application so that it (a) has the required cores for all of its tasks and (b) satisfies its strict deadline is a significantly harder task than the embedding of either of the other three applications. Furthermore, in the presented experiment, a greater success during the embedding of the automotive application implies that on average, the strategy provided by LOCAL is applied to more difficult deployment situations where the platform is already occupied by a higher number of instances of the automotive application.

The fact that the strategy of LOCAL consists of 80 OPs, as opposed to the 400 OPs generated by DAARM, constitutes an additional advantage, as the size of the strategy directly impacts the computational power requirements of the RTM. Indeed, with the assumption of equal deadlines imposed on the decision process of the RTM, the strategy provided by LOCAL not only offers a better embeddability, but also requires a RTM with a 5× lower computational power. The better performance of LOCAL can be explained by its characteristic features: Instead of trying to guide the optimizer by artificial objectives, the run-time scenario is recreated within the optimization framework, which adapts to the problem and autonomously develops a strategy considering various OCs.

V. Conclusions

In this paper, we have proposed Learning Optimizer Constrained by ALtering conditions (LOCAL), a novel framework for the optimization of dynamic problems, which was developed for the automatic design of adaptive embedded systems. Instead of a single solution, LOCAL optimizes a strategy, i.e., a set of solutions, each of which is to be used under different operational conditions. The strategy is optimized in an iterative interaction with a so-called environment. The presented experimental results provide evidence that the expressiveness of the LOCAL framework enables a straightforward modeling of the complex embeddability objective,

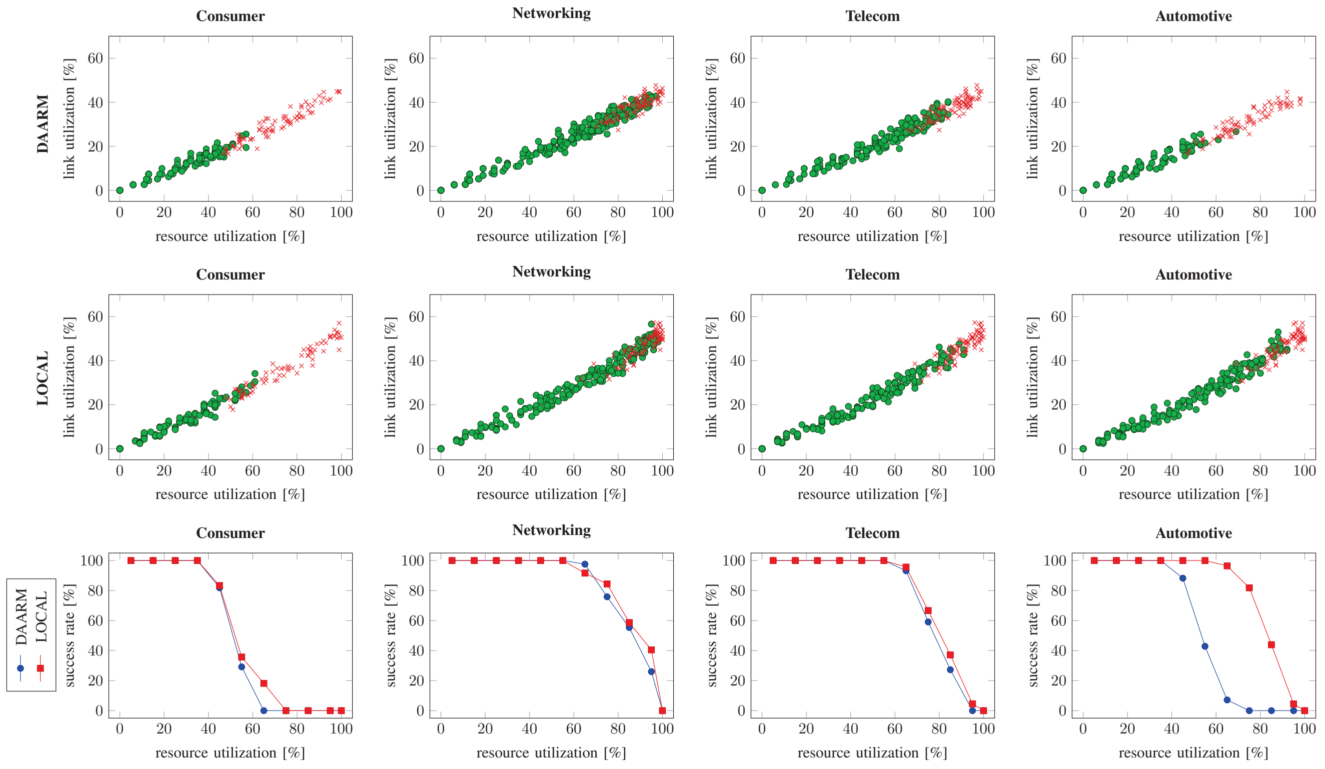

Fig. 3. Scatter plots of the deployment attempts (green dot: success, red cross: failure) of the two approaches observed throughout the experiment (first and second row) and an illustration of the overall success rate (third row). The columns correspond to the four applications from the E3S benchmark suite [5].

outperforming a s.o.t.a. approach which is specifically tailored for embeddability optimization.

References

[1] E. W. Brião, D. Barcelos, et al. Dynamic task allocation strategies in mpsoc for soft real-time applications. In *DATE*. 2008.
[2] C. Bu, W. Luo, et al. Continuous dynamic constrained optimization with ensemble of locating and tracking feasible regions strategies. *IEEE TEVC*, 2016.
[3] E. L. de Souza Carvalho, N. L. V. Calazans, et al. Dynamic task mapping for mpsocs. *IEEE Design & Test of Computers*, 2010.
[4] K. Deb, A. Pratap, et al. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Transactions on Evolutionary Computation*, 2002.
[5] R. Dick. Embedded system synthesis benchmarks suite (E3S), 2010.
[6] M. Greeff and A. P. Engelbrecht. Solving dynamic multi-objective problems with vector evaluated particle swarm optimisation. In *CEC*. 2008.
[7] M. Heibig, H. Zille, et al. Performance of dynamic algorithms on the dynamic distance minimization problem. In *GECCO*. 2019.
[8] J. Holland and J. Reitman. Cognitive systems based on adaptive agents, 1978.
[9] D. Le Berre and A. Parrain. The sat4j library. *JSAT*, 2010.
[10] M. Lukasiewycz, M. Glaß, et al. SAT-decoding in evolutionary algorithms for discrete constrained optimization problems. In *CEC*. 2007.
[11] —. Opt4J - A Modular Framework for Meta-heuristic Optimization. In *GECCO*. 2011.
[12] M. Lukasiewycz, S. Shreejith, et al. System simulation and optimization using reconfigurable hardware. In *Proceedings of ISIC*. 2014.
[13] M. Maggio, H. Hoffmann, et al. Comparison of decision-making strategies for self-optimization in autonomic computing systems. *ACM TAAS*, 7(4):36, 2012.
[14] K. Pal, C. Saha, et al. Differential evolution and offspring repair method based dynamic constrained optimization. In *SEMCCO*. 2013.
[15] J. Panerati, F. Sironi, et al. On self-adaptive resource allocation through reinforcement learning. In *AHS*, pp. 23–30. IEEE, 2013.
[16] B. Pourmohseni, F. Smirnov, et al. Isolation-aware timing analysis and design space exploration for predictable and composable many-core systems. In *ECRTS*, pp. 12:1–24. 2019.
[17] F. Reimann, M. Lukasiewycz, et al. OpenDSE – open design space exploration framework, 2018.
[18] N. Satish, K. Ravindran, et al. A decomposition-based constraint optimization approach for statically scheduling task graphs with communication delays to multiprocessors. In *DATE*. 2007.
[19] H. Shojaei, A. Ghamarian, et al. A parameterized compositional multi-dimensional multiple-choice knapsack heuristic for cmp run-time management. In *DAC*.
[20] A. K. Singh, M. Shafique, et al. Mapping on multi/many-core systems: survey of current and emerging trends. In *DAC*. 2013.
[21] F. Smirnov, B. Pourmohseni, et al. IGOR, Get Me the Optimum! Prioritizing Important Design Decisions During the DSE of Embedded Systems. *TECS*, 18(5s):78, 2019.
[22] S. F. Smith. A learning system based on genetic adaptive algorithms. 1980.
[23] L. Thiele, L. Schor, et al. Thermal-aware system analysis and software synthesis for embedded multi-processors. In *DAC*. 2011.
[24] R. J. Urbanowicz and J. H. Moore. Learning classifier systems: a complete introduction, review, and roadmap. *Journal of Artificial Evolution and Applications*, 2009.
[25] A. Weichslgartner, D. Gangadharan, et al. Daarm: Design-time application analysis and run-time mapping for predictable execution in many-core systems. In *CODES+ISSS*. 2014.
[26] S. W. Wilson. Zcs: A zeroth level classifier system. *Evolutionary computation*, 1994.
[27] —. Classifier fitness based on accuracy. *Evolutionary computation*, 1995.
[28] P. T. Wolkotte, G. J. Smit, et al. Energy model of networks-on-chip and a bus. In *SoC*. 2005.
[29] J. Zeppenfeld, A. Bouajila, et al. Learning classifier tables for autonomic systems on chip. *INFORMATIK 2008. Beherrschbare Systeme-dank Informatik. Band 2*, 2008.
[30] C. Zhu, Z. P. Gu, et al. Reliable multiprocessor system-on-chip synthesis. In *CODES+ISSS*. 2007.