Hybrid Adaptation Policies – Towards a Framework for Classification and Modelling of Different Combinations of Adaptation Policies

Frank Trollmann
DAI-Labor, TU Berlin
10587 Berlin
Germany
frank.trollmann@dai-labor.de

Johannes Fähndrich
DAI-Labor, TU Berlin
10587 Berlin
Germany
johannes.fähndrich@dai-labor.de

Sahin Albayrak
DAI-Labor, TU Berlin
10587 Berlin
Germany
sahin.albayrak @dai-labor.de

ABSTRACT

The development[1] of self-adaptive systems can greatly benefit from reference frameworks to structure the development process. Current reference frameworks abstract from the adaptation decision – the selection of a specific adaptation based on the goals and options available. This decision is usually based on the three adaptation policies: rule-based, goal-based or utility based. However, concepts from these policies often are combined with each other in different ways to achieve hybrid policies. To structure this combination this paper identifies four types of such combinations. We express these types as patterns within a model that captures the functions, models and relations participating in the adaptation decision.

CCS CONCEPTS

• **Software and its engineering** → Software organization and properties → Software system structures → Abstraction, modeling and modularity

KEYWORDS

Adaptation Decision, Adaptation Policies, Reference Model

1 INTRODUCTION

Researchers have aimed to provide structured frameworks and guidelines for the implementation of self-adaptive systems. Examples are the MAPE-K loop [5, 11], which has been implemented and extended in several ways, and FORMS [24], which provides a formal underpinning for modelling the computations and information participating in a self-adaptive system (SAS). These frameworks enable structured engineering of self-adaptive systems, comparison of approaches and reuse of concepts and implementations.

One of the key aspects of a self-adaptive system is the adaptation decision: finding an appropriate adaptation given the current situation and goals. While existing frameworks usually define a component or function to encompass this decision, the implementation of the decision making is not detailed and left to the developer. However, since the adaptation decision is one of the central aspects of a self-adaptive system, developers could significantly benefit from structures and guidelines.

To represent different approaches to implement adaptation decisions, Kephard and Walsh use adaptation policies from artificial intelligence [12]. They classify policies as rule-based (deriving adaptations from condition-action rules), goal-based (searching for adaptations to fulfil a goal) and utility-based (searching adaptations that maximize a utility function). These policies are often combined with each other [8]. We call those combinations hybrid adaptation policies.

In this paper we present our classification of four types of combination of adaptation policies that lead to hybrid policies. To characterize these types we introduce a model to represent computations, information and relations involved in the adaptation decision and identify patterns in this model that correspond to the types of hybrid policies. This represents a first step towards a structured framework for modelling, classifying and comparing approaches for making adaptation decisions.

In the following section, we review state of the art in self-adaptive systems concerning adaptation decisions. Section 3 describes two running examples that will be used for illustration. The foundation of the paper is given in Section 4. It consists of category theory, as the foundation of our modelling notation, and a description of the three adaptation policies. Section 5 classifies hybrid policies and describes them as model patterns. These are applied to the running examples in Section 6. Section 7 discusses limitations and extensions planned in future work.

2 STATE OF THE ART

This section considers existing reference frameworks to derive information relevant to classifying adaptation decisions.

Several frameworks discuss the relation between multiple MAPE-K loops. They deal with when and how multiple loops interact [22, 23]. Those approaches do not detail the plan phase of the original MAPE-K loop. However, they give an idea on how multiple decisions, distributed over multiple loops, can interact. The DYNAMICO framework splits the adaptation up into three

SEAMS '18, May 28–29, 2018, Gothenburg, Sweden
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5715-9/18/05…$15.00
https://doi.org/10.1145/3194133.3194137

feedback loops [18]. The classic adaptation feedback loop is accompanied by a feedback loop concerning the monitoring infrastructure and a feedback loop concerning goals. While these feedback loops still abstract from how the plan phase is implemented, the goal feedback loop involves maintaining, reasoning about and changing an explicit notion of goal. Such a goal representation is also necessary for reasoning about goal fulfilment in the adaptation decision.

Vogel et al. extend the MAPE-K loop from a model driven engineering perspective in their EUREMA framework [20, 21]. Here, the knowledge base is divided into models. EUREMA distinguishes between reflective models, which represent the running system, an evaluation model, which represents the adaptation goals, and a change model, which represents the adaptation options available to the plan phase. While the approach does not detail the plan phase, it structures its input by providing explicit models for goals (evaluation model) and adaptation options (change model).

FORMS is a formal framework for architecture choices in self-adaptive systems [24]. In this framework a self-adaptive system is composed of subsystems, distinguished into base-level subsystems (the adaptive system itself) and reflective subsystems (that monitor, represent or change other software systems). The framework enables expressing models and computations used by those subsystems. The adaptation decision is a computation in reflective subsystems. FORMS is aligned with the MAPE-K loop and distinguishes reflective computations for its four phases. It does not detail how these computations are implemented.

Hussein et al. classify adaptation mechanisms with adaptation policies [9]. The three adaptation policies have been inspired from artificial intelligence where they have been defined as action selection policies for artificial agents [15]. They have been applied to the self-adaptive systems domain earlier by Kephart and Welsh [12] and Balasubramanian et al. [1]. During their classification, Hussein et al. notice that in some cases multiple adaptation policies are mixed and predict this as a future trend for self-adaptive systems.

To verify this trend we performed a structured literature analysis. The purpose of this analysis was to get a representative sample of existing approaches and determine whether they require hybrid adaptation policies. To retrieve a representative sample set we focused on the SEAMS conference in the years 2016 [26] and 2017 [27][2]. For classification we used the distinction between evaluation and change model made in EUREMA [21]. The evaluation model describes which information is used to represent the goals of the adaptation and the change model describes which options are available for adaptation. For each model we identified elements from the three adaptation policies (cf. Section 4.2). The result is shown in Fig. 1. Each dot represents a set of approaches with the same properties. The number in the dot represents the number of approaches. The location of the dot represents the combination of evaluation and change model. Dots on the line between two categories represent that elements from both categories are present. For example, the lighter dot represents that there are two approaches with a goal-based change model and elements from goal- and utility-based evaluation models.

These findings confirm that adaptation policies are frequently mixed. Of the 26 approaches only seven use pure adaptation policies. The other approaches either combine evaluation and change model of different policies or use elements from multiple policies in their evaluation/change model (dots on the lines between two categories).

The literature review also revealed multiple ways in which these policies are combined. Thus, even approaches that are represented by the same dot In Fig. 1 may differ in how they combine the elements from the respective policy. This paper classifies these types of combination in four general types of hybrid policies, based on the experiences of the structured literature review. In the next section we will use two running examples from the literature review to illustrate two different types of combination.

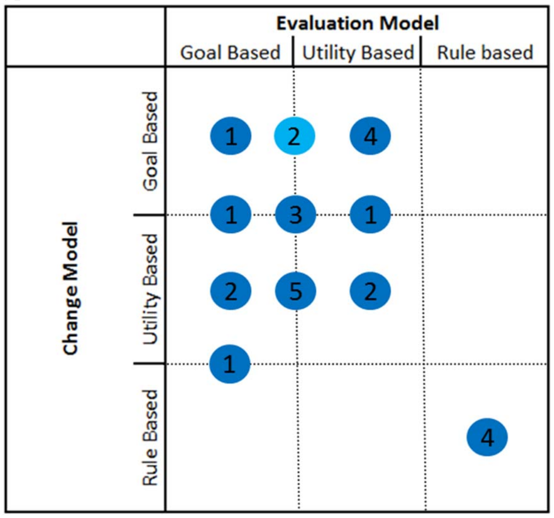

Figure 1: Classification of adaptation policies published in the last years 2016 and 2017 at the SEAMS conference.

3 RUNNING EXAMPLES

In this section we present two examples of self-adaptive systems. These examples illustrate that there are different kinds of combination of adaptation policies and serve as running example for the classification and model presented in Section 5.

The first example is **DeltaIOT**, an exemplar for adaptation in the Internet of Things that aims to enable researchers to evaluate and compare their approaches [10]. DeltaIOT consists of a multihop network with 25 communication nodes and various sensors based on LORA communication. The goal of adaptation is to maintain connectivity of each node to a central gateway. This goal is accompanied by the following side-constraints:

- *Reduce Package Loss* by guaranteeing a bounded number of packets lost in the whole network
- *Minimise energy consumption* over the whole network

[2] Detailed results are provided on: https://figshare.com/s/2bfb135bcb1920c15b4b.

- *Minimise packet loss* over the whole network
- *Balance energy consumption* by minimising imbalance

Adaptation is required due to uncertainty. Examples are interference in the wireless network or unavailability of nodes. To react to these situations, an adaptation approach can change the network settings for the connection between neighbouring nodes. These settings consist of three numeric parameters: transmission power, spreading factor and distribution factor.

Iftikhar et al provide an example adaptation mechanism for wireless interference and fluctuating traffic load [10]. The example approach uses a MAPE-K loop to monitor network parameters and, in case they are not optimal, plan a reaction. This is done by modifying the transmission power and the distribution factor gradually over multiple iterations of the MAPE-K loop until the network parameters are optimal.

The second example is the learning and evolution approach in dynamic software product lines (DSPL) by Sharifloo et al. [16]. We call this example **L-DSPL**. DSPL use rule-based adaptation [7]. A feature-model describes features of the software system and contains variability, e.g., by having alternate components for the same role. Adaptation rules select the configuration of features based on context situations. Learning and evolution is motivated by the uncertainty present when defining the adaptation rules.

Learning updates the adaptation rules to deal with unexpected context changes or situations in which the system exhibits non-optimal behaviour. For this purpose, the authors add an explicit notion of requirement that can be evaluated at run time. A situation that does not meet the requirements triggers the learning process. Possible actions for learning are to change, remove and add adaptation rules. Learning cannot change the feature-model.

The evolution process is responsible for evolving the feature model. In this process unused features may be removed or new features may be added to address a situation that cannot otherwise be addressed. Adding new features requires extending the software system. Thus, the evolution process is executed by the developer (e.g., during maintenance). Since the evolution process is not coordinated by an automated software system at runtime we focus on the DSPL and learning processes in this paper and exclude the evolution process.

Both examples represent hybrids of two adaptation policies. DeltaIOT contains goals from goal- and utility-based adaptation policies. Node connectivity, a Boolean goal, is accompanied by utility functions, like energy consumption. L-DSPL implements a rule-based adaptation policy, combined with a goal function and actions, which are elements of goal-based policies.

The two approaches are also fundamentally different in how they combine adaptation policies. DeltaIOT has a single adaptation mechanism that can take into account both Boolean goal descriptions as well as optimization functions. L-DSPL, on the other hand, has two separate adaptation mechanisms: the DSPL approach, which is responsible for adapting the software application, and the learning component, which is responsible for adapting the rules in the DSPL approach.

4 FOUNDATIONS

This section describes the foundations for understanding the categorization of hybrid policies. As formal foundation we chose category theory. Category theory and its interpretation as representation of adaptive systems are introduced in Section 4.1. Section 4.2 describes the three adaptation policies.

4.1 Category Theory

Category theory is a formal framework that focuses on the interrelation of objects [6]. In this paper we use category theory to formalize an adaptive system by assuming that the adaptive system is represented by a category with certain properties. This notion also lets us abstract from any specific adaptive system as we will only need to assume that a category with the respective properties exists and do not have to know the specific category.

A category consists of four elements $(Ob, Mor, ^\circ, id)$. Ob is the set objects contained in the category and Mor is a set of morphisms between those objects. Morphisms can be imagined as directed arrows, each connecting two objects. These arrows usually represent specific relations between objects that can differ from category to category. For example, in a category of sets of Latin letters the objects can represent sets such as $\{A\}$, $\{A, G\}$ or $\{A, G, Z\}$. Morphisms represent the subset relation, meaning the set contains arrows such as $\{A\} \xrightarrow{\subseteq} \{A, G\}$. A category further requires two properties. It requires that morphisms can be composed sequentially by an operator $^\circ$ and that identical morphisms id exist for each object. In our example both properties are fulfilled because the subset relation is transitive (e.g., $\{A\} \xrightarrow{\subseteq} \{A, G\}$ and $\{A, G\} \xrightarrow{\subseteq} \{A, G, Z\}$ imply $\{A\} \xrightarrow{\subseteq} \{A, G, Z\}$) and reflexive (e.g., $\{A\} \xrightarrow{\subseteq} \{A\}$ exists).

Categories have been used before to represent models of adaptive or dynamic systems (e.g., in [17]). Objects represent models of the language (e.g., all class diagrams) and morphisms represent relations between them. This enables talking about models and their relation without referring to a specific modelling language. This has for example been used in graph transformation to prove results that hold for all modelling languages with certain properties [4].

We use categories to represent adaptive systems. This is inspired by Zhang et al., who view an adaptive program as a set of steady state programs, which are "*non-adaptive program(s) suited for a specific set of environmental conditions*" [25]. Adaptation is a transition between steady state programs. In our category representation the objects correspond to steady state programs and morphisms represent adaptations between them.

An adaptation is not the only possible change. Adaptations are usually a reaction to the dynamic behaviour of the software system and its environment. These dynamic behaviour changes can encompass state changes of the adaptive system itself, e.g., its execution state, or changes in monitored context variables. To be able to capture all of these changes we extend our notion of a category representing an adaptive system. We define objects to contain information about the adaptive system as well as its environment. Morphisms can represent any change, including changes caused by dynamic behaviour and changes caused by adaptation. In some places we will need to distinguish between both types of changes. For doing so we require that the dynamic behaviour and adaptation behaviour each form a subcategory of the adaptive system representation which only represents the respective changes. This is expressed in the following definition:

Definition 1 (Adaptive System Representation). *An adaptive system is represented by a category $S = (Ob_S, Mor_S, \circ_S, id_S)$ whose objects Ob_S represent all possible states of the adaptive system and its' environment and whose morphisms represent changes to the self-adaptive system or its environment.*
The adaptation behaviour of an adaptive system S is represented by a category $Ad_S = (Ob_S,\ Mor_S^{Ad} \subseteq Mor_S, \circ_S, id_S\ )$ whose morphisms Mor_S^{Ad} represent all adaptations.
The dynamic behaviour of an adaptive system S is represented by a category $Beh_S = (Ob_S,\ Mor_S^{Beh} \subseteq Mor_S, \circ_S, id_S\ )$ whose morphisms Mor_S^{Beh} represent all monitored changes to the self-adaptive system or its environment.

Using this definition we can talk about changes of a SAS in general, by referring to Mor_S, or refer to adaptations or dynamic behaviour exclusively by referring to Mor_S^{Ad} or Mor_S^{Beh} respectively. The reader should be aware that due to the properties of a category all three types of changes contain identities and sequential composition of changes. Thus, they are able to express non-changes, by using identities, and a sequence of changes is also a valid change, due to sequential composition.

To align this view on adaptive systems with more traditional views on self-adaptive systems we integrate it into the MAPE-K loop. For this, we use the extension of the MAPE-K loop proposed by Vogel et al. [20]. The integration is shown in Fig. 2. As stated in Section 2, this framework provides three types of models: the *evaluation model*, representing the goals of an adaptation, the *change model*, representing how the system can be changed, and the *system representation*, representing the state of the self-adaptive system and its environment.

We assume that the system representation is an adaptive system representation (cf. Definition 1). The system representation is a direct representation of the adaptive system managed by the MAPE-K loop. Vogel et al. further classify the system representation (called reflection models) into system models, representing the running system, environment models, representing the environment and analysis models, representing information derived via analysis. Our system representation is on the level of general reflective models as our adaptive system representation encompasses the system and its environment. Since it assumes an arbitrary category it may contain specialized submodels, if required by a specific approach.

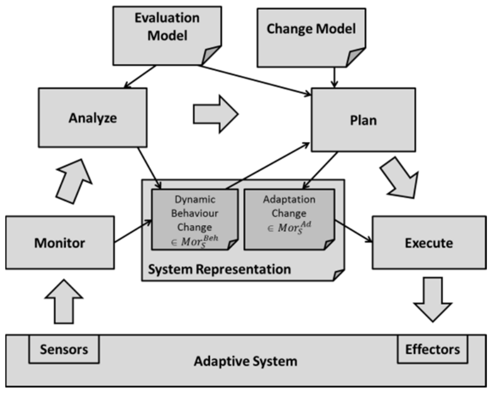

Figure 2: Relation of the formalization to the MAPE-K loop extension developed by Vogel et al. [20].

In the MAPE-K loop *Monitor* and *Analyze* update the system representation. The produced update is a morphism from the dynamic behaviour of the self-adaptive system. *Plan* implements an adaptation policy that derives an adaptation. It uses the dynamic behaviour change, evaluation model and change model as input. The produced adaptation is a morphism from the adaptation behaviour of the adaptive system. *Execute* is responsible for implementing this change.

This paper focuses on how adaptation policies derive the adaptation in the plan phase. The search space of an adaptation policy is usually restricted by the goal it aims to fulfil and the adaptations it can apply in a given situation. We formulate this information as adaptation problem. Based on the terminology from Fig. 2, we define the adaptation problem as follows:

Definition 2 (Adaptation Problem). *Given two categories representing the change model (CM) and evaluation model (EM), an adaptation problem is a four-tuple (cm, em, conforms to, fulfils) where $cm \in Ob_{CM}$ is a change model, $em \in Ob_{EM}$ is an evaluation model, $conforms\ to \subseteq Ob_{CM} \times Mor_S^{Beh} \times Mor_S^{Ad}$ is a relation specifying the conformity of an adaptation with the change model and $fulfils \subseteq Ob_{EM} \times Mor_S^{Ad}$ is a relation specifying when an adaptation fulfils the evaluation model.*

The adaptation problem contains the evaluation and change model and relations that specify how they are applied. The evaluation model *em* defines the goal of the adaptation. The relation $fulfils$ describes how this goal is evaluated. For example, if the evaluation model is a double-valued function the relation *fulfils* describes whether an adaptation is required to minimize or maximize this function. The change model *cm* describes which adaptations are considered valid. The relation *conforms to* is used to evaluate whether an adaptation adheres to this model. For example, the change model could consist of a set of actions that can be applied and the *conforms to* relationship could test whether an adaptation can be achieved by applying these actions to the current state of the adaptive system. Thus, overall the adaptation problem specifies which options are available for adaptation (*cm, conforms* to) and how the success of an adaptation is evaluated (*em, fulfils*).

In Section 5 we will use the adaptive system, adaptation problem and plan phase as means to classify and model hybrid adaptation approaches.

4.2 Adaptation Policies

In this section we introduce the three adaptation policies. To capture their differences we discuss differences in their adaptation problem (Definition 2) and plan phase. The reader should be aware that the policies described here are stereotypical, i.e., actual approaches may choose to implement details differently or add additional elements depending on the requirements of a specific approach. Fig. 3 depicts an overview of the relevant information.

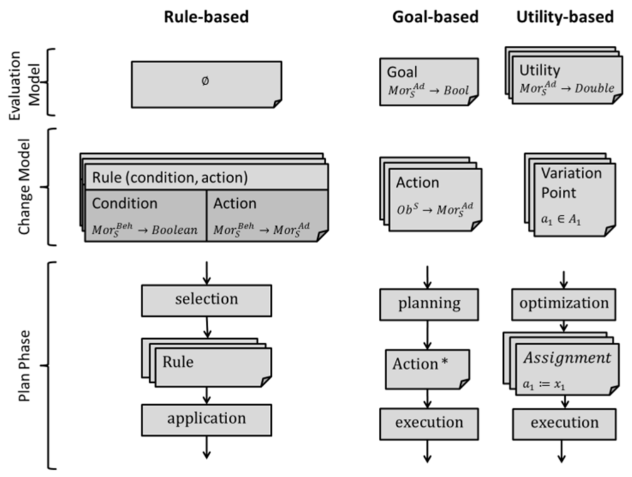

Figure 3: Pure Adaptation Policies.

In the **rule-based policy** the change model consists of a set of adaptation rules. Each rule consists of a condition, which specifies whether the rule should be applied based on the dynamic behaviour change, and an action, which applies the rule. This policy does not contain an explicit evaluation model. This reflects the fact that the goal has been used to define the rules and thus is implicit in their conditions and actions and not formalized separately. Since there is no evaluation model the relation *fulfils* has no information to restrict possible adaptations and is defined as true (always fulfilled). The plan phase consists of two functions. *selection* selects a set of rules to apply based on their condition. The function *application* applies the rules and resolves conflicts (e.g., there are several notions of conflict and dependency in graph transformation [2]).

The dynamic software product line in L-DSPL is an example of a rule-based policy. If we exclude the learning component the DSPL approach consists of a set of rules, which select the right feature model. These rules have mutually exclusive conditions based on the current context situation. This means, the result of *selection* is exactly one rule. The action substitutes the current feature model with the one prescribed by the rule. Thus, *application* applies the selected rule, if it is not applied already.

The rule-based policy can be implemented and extended in different ways. E.g., in L-DSPL only one rule can be applied at a time, whereas other approaches could allow the application of multiple rules. The two functions in the plan phase can also be implemented in different ways, e.g., prioritizing certain rules or using conflict notions on different levels of granularity [3]). Actions can also be extended with more information, e.g., by requiring the action to be reversible.

As suggested by the name, the **goal-based policy** is governed by a goal function in its evaluation model. This function is a Boolean function that determines if the adaptation fulfils the goal. The change model consists of a set of actions that can be used to reach the goal. Each action is a function that can be applied to derive an adaptation. The plan phase consists of a function *planning*, which selects a sequence of actions and a function *execution*, which executes them to derive an adaptation.

The learning part of L-DSPL is an example for a goal-based policy. This part optimizes the rules of the DSPL approach by using actions that add rules, remove rules and change rules. The actions are carried out to ensure the DSPL approach fulfils all requirements (= Boolean goal function). An example for such a requirement is for the rules to be unique and complete, meaning in every possible context situation exactly one rule is applied.

Again, there are multiple ways to implement this policy. E.g., the goal may refer to the result of the adaptation or to the adaptation itself. Actions may also be extended, e.g., by applicability conditions.

In the **utility-based policy,** the evaluation model contains one or more utility functions that map an adaptation to a double value. The change model identifies a set of variation points and intervals for their possible values. The plan phase can search within those intervals for an optimal combination. It consists of a function *optimization*, which determines the optimal values for the variation points and a function *execution* which implements these values and derives the according adaptation.

DeltaIOT is a combination of goal- and utility-based policies. If we ignore the Boolean goals (connectivity, sums of connections is 100, reduce package loss), the remaining parts represent a utility-based policy. The utility function is composed of three subfuctions minimise energy consumption, minimise package loss and balance energy consumption. The network parameters of each node represent the variation points. Here, each contained parameter has to be within a specific interval (transmission power: $\{-3, \dots 18\}$, spreading factor: $\{7, \dots 12\}$, distribution factor: $\{0, \dots 100\}$).

Different ways to implement the utility-based policy concern different types of variation points (e.g, choosing integer parameters, configuring an architecture model, ...) and different types of utility function. In general, the utility function may refer to the result of the adaptation (e.g., the energy consumption of a configuration in DeltaIOT) or to the adaptation itself (e.g., the cost or benefit of the change). Depending on the function, minimization or maximization may be required by *fulfils*. If the evaluation model contains multiple utility functions the relation *fulfils* also has to specify how they relate (tradeoffs, requirement of pareto-optimality,…)

Our utility-based policy differs from the traditional utility-based policy [15], which is an extension of the goal-based policy with a utility function. The description above does not contain actions, but a set of variation points. The purpose is to have three prototypical policies which are as different from each other as possible. This is inspired by the difference between the planning and optimization problem in artificial intelligence. From this point of view, planning with a utility function represents a hybrid of the goal- and utility-based policy.

To further illustrate the difference between these two policies we use a short example. Our goal based policy searches a way through a two-dimensional grid while avoiding obstacles. It can move up, down, left or right (actions). Our utility-based policy searches the maximum of a function with two integer parameters. While both policies search a combination of two integer parameters, the goal-based policy traverses the state space one square at a time while the utility-based policy can try combinations of parameters in any order to find an optimum.

Summarizing, each of the three pure adaptation policies has its stereotypical version of the evaluation and change model. While in praxis their implementation can vary according to the specifics of a solution algorithm, these models are usually very similar, or are extensions of the ones we present here.

5 CLASSIFICATION

In this Section we present our classification of ways to create hybrid policies from the adaptation policies from Section 4.2. The classification is based on our experiences with the structured literature review, described in Section 2. For the classification we observe how the combined policies relate in their adaptive systems, adaptation problem and plan phase.

Since adaptation policies can only be combined if they concern the same or related adaptive systems we use the adaptive system as primary classification criteria. Two policies can be related in their adaptive system in the following ways. Their adaptive systems can be the *same* or different. If they differ they may still be related in the sense that they are part of a *common parent* system, e.g., they are adapting specific concerns of a larger adaptive system. Two policies can also be related if the adaptive system of one is *information used by the other policy*, e.g., in L-DSPL, where the rules in the change model of the DSPL approach are adapted by the learning mechanism. If neither of these cases apply we say the adaptive systems are *unrelated*.

If two policies target the same adaptive system we classify them with respect to relations of their adaptation problem. They may be *integrated*, meaning that components from both adaptation problems are combined into one adaptation problem that is solved within a single MAPE-K loop, or *separate*, meaning there are multiple adaptation problems and MAPE-K loops.

If the adaptive system is the same and the adaptation problem is integrated the combined policies adapt the same system with the same purpose in one MAPE-K loop. In this situation we can distinguish two cases based on whether the overall plan phase keeps the plan phases of the compined policies *separate*, e.g., by calling them from the outer plan phase and merging/selecting their results. If the plan phases from the combined policies cannot be distinguished we say they are *integrated*.

Based on these three criteria, Fig. 4 illustrates the classification of types of combination of adaptation policies. In this classification we identify four types of combination: integrated, coordinated, concurrent and hierarchical.

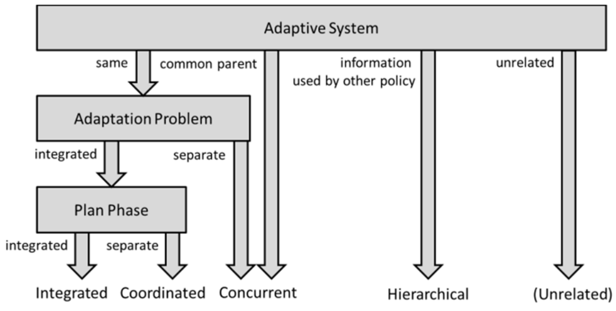

Figure 4: Classification of ways to create hybrid policies.

In the next subsection we will introduce the concrete syntax of a modeling notation to help us to represent these four types of hybrid policies in terms of the involved computation, relations and models. We will use this notation to model and explain the four types of combination in the following subsections.

5.1 Model Notation

To illustrate the different types of combinations of adaptation policies we use a modelling notation. We need to represent functions, how they are related in terms of information and how they relate to relations like *conforms to* and *fulfils*.

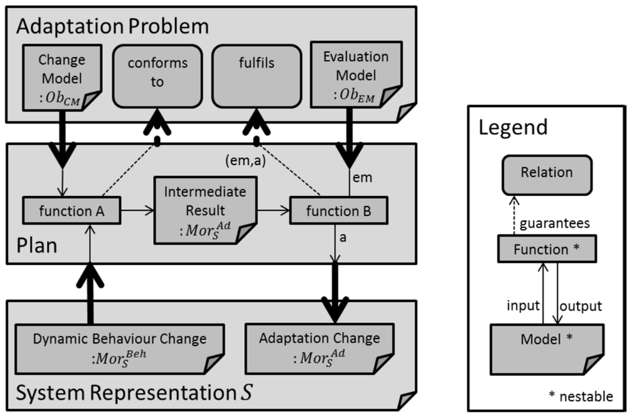
Adaptation Problem
Change Model $:Ob_{CM}$
conforms to
fulfils
Evaluation Model $:Ob_{EM}$
(em,a)
em
function A
Intermediate Result $:Mor_S^{Ad}$
function B
a
Plan
Dynamic Behaviour Change $:Mor_S^{Beh}$
Adaptation Change $:Mor_S^{Ad}$
System Representation S
Legend
Relation
guarantees
Function *
input
output
Model *
* nestable

Figure 5: Visual notation of adaptation policies.

This notation builds on the foundations from Section 4. Specifically, we assume that for each model there is an adaptive system representation (Definition 1) that denotes the modelling language of this model. Models are either objects or Morphisms from this category. This enables us to represent information that may change due to dynamic run time behaviour or adaptation. Specifically, we will make use of this for defining the hierarchical adaptation policy. If a model is not adaptive or cannot change, the respective set of morphisms in the modelling language can be defined as empty.

Fig. 5 shows the overview of the notation. It contains a legend (right hand part) and an example of a plan phase using this notation (left hand part). The notation consists of models, functions and relations. Models contain information and represent the connection to our category-theoretic foundation.

Functions are related to models by two arrows indicating that they use a model (*input*) or produce a model (*output*). Functions are related to relations by dashed arrows, indicating that they guarantee that their input and output fulfil the relation. For example, $function\,A\colon Ob_{CM} \times Mor_S^{Beh} \rightarrow Mor_S^{Ad}$ fulfils the relation $conforms\,to \subseteq Ob_{CM} \times Mor_S^{Beh} \times Mor_S^{Ad}$, meaning for $function\,A(cm, beh) = ad$ we have $conforms\,to\,(cm, beh, ad)$. In cases where the correlation based on types is ambiguous we use annotations on arrows. E.g., the annotations on *function B* denote that it is the output adaptation that is related to the evaluation model by relation *fulfils*, not the input adaptation.

Both functions and models can nest elements. Functions can be implemented via a workflow of subfunctions and models can contain more fine-granular information.

The left part of Fig. 5 shows how we represent the plan phase of the MAPE-K loop. The plan phase is defined to have the input and output specified in Fig. 2. It uses the adaptation problem (cf. Definition 2) thus has access to the *evaluation model* and *change model* and fulfils relations *conforms to* and *fulfils*. It is connected to the system representation, which uses an adaptive system representation (cf. Definition 1) as modelling language. From this representation the plan phase observes a dynamic behaviour change and produces an adaptation change. It fulfils the relations *conforms to* and *fulfils*. In the figure, the content of the plan phase is exemplified via two inner functions, *function A* and *function B*, producing an intermediate adaptation.

The reader should note that this notation is not intended as complete modelling notation, despite its formal grounding. E.g., it lacks clear execution semantics, like the relation between nested elements. For the purpose of this paper this is sufficient since we use it to illustrate the difference between hybrid policy types. To be usable as framework the respective execution semantics need to be provided, e.g., by a mapping into EUREMA [21] or process-based notations such as BPMN [13]. In this paper, we take certain liberties with the notation. For reasons of readability we will use textual descriptions in quotation marks to abbreviate relations whose formal definition is too unwieldy.

5.2 Integrated Combination

An integrated hybrid adaptation policy combines multiple policies targeting the same adaptive system and adaptation problem with an integrated plan phase.

An example is the classical interpretation of the utility-based policy, depicted in Fig. 6. In this policy the goal-based policy is combined with a utility function. This utility function is added to the evaluation model. Accordingly, the combined evaluation model consists of a tuple of evaluation models, one from each policy. The relation *fulfils* is also extended to check whether the adaptation fulfils the goal and optimizes the utility function. Formally, this means, the relation is an intersection of the relations *fulfils* of both policies. In the plan phase function $optimize\_with\_actions$ (which is able to plan action application with a utility function) is used to derive the required adaptation.

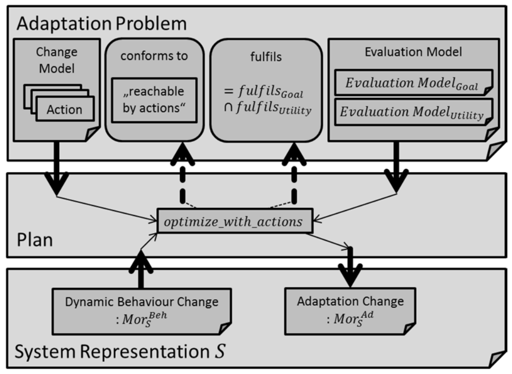

Figure 6: Illustration of an integrated hybrid policy.

In our modelling notation an integrated combination can be recognized by the fact that the adaptation problem, plan phase and adaptive system are fully merged. This means, there is a single integrated plan phase targeting one adaptive system and an integrated adaptation problem. The integration of adaptation problems can be based on the evaluation model, change model or both as well as the respective relations. The plan phase in this policy is integrated, meaning it is not a composition of the separate plan phases of the respective policies but is implemented in terms of its own functions. In the example, neither the pure goal-based nor the utility-based policy contains a function $optimize\_with\_actions$ that is able to optimize a utility function via application of actions.

5.3 Coordinated Combination

A coordinated hybrid adaptation policy combines multiple adaptation approaches in one MAPE-K loop but keeps their plan phases separate. This can be modelled as an outer plan phase in which the plan phases of the combined policies are called.

Fig. 7 gives an example in which two plan phases, $Plan_A$ and $Plan_B$, are nested within one outer plan phase. Each inner plan phase is called with the original adaptation problem and the dynamic behaviour change. Each plan phase produces a separate result (Res_A and Res_B). A function *select* selects the result that best fulfils the utility function.

Like the integrated combination, the coordinated combination contains one adaptive system and adaptation problem. The plan phase, however, can be subdivided into the original plan phases from the combined policies. As illustrated by *select,* the outer plan phase may contain additional functions, e.g., to split up the adaptation problem into subproblems or to combine the result. The example in Fig. 7 represents an ensemble in which multiple policies are executed and the best one is selected. Other combinations could aim to find the intersection of two sub-problems or to find an adaptation that contains both results.

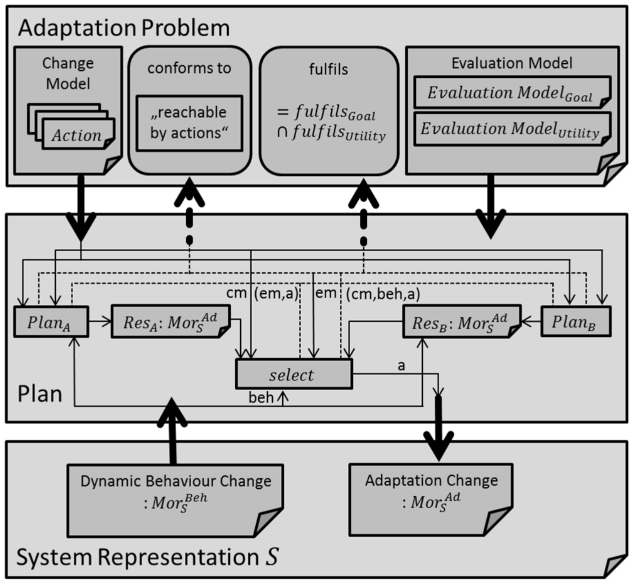
Adaptation Problem
Change Model
Action
conforms to
„reachable by actions"
fulfils
$= fulfils_{Goal} \cap fulfils_{Utility}$
Evaluation Model
$Evaluation\ Model_{Goal}$
$Evaluation\ Model_{Utility}$
cm (em,a) em (cm,beh,a)
$Plan_A$
$Res_A: Mor_S^{Ad}$
$Res_B: Mor_S^{Ad}$
$Plan_B$
select
a
beh
Plan
Dynamic Behaviour Change $: Mor_S^{Beh}$
Adaptation Change $: Mor_S^{Ad}$
System Representation S

Figure 7: Illustration of a coordinated hybrid policy.

5.4 Concurrent Combination

In a concurrent hybrid policy the combined policies solve different adaptation problems, in different MAPE-K loops, but target the same adaptive system directly or indirectly.

This is illustrated in Fig. 8. In the *direct* case (a), two plan phases target the same adaptive system. This means, they operate on the same category, representing the adaptive system. However, the distinction between dynamic behaviour and adaptation may be different. E.g., the adaptation caused by one plan phase may be considered dynamic behaviour by the other plan phase.

In the *indirect* case (b) the adaptive system of both plan phases is part of a common overall adaptive system. This means, there is an adaptive system that encompasses both adaptive systems S_A and S_B. Even though both adaptations target different adaptive system they may still influence each other.

In our model the concurrent case is signified by separate plan phases that adapt the same adaptive system, either by having this system as target of the two plan phases directly or by indirectly targeting two (possibly overlapping) subsystems.

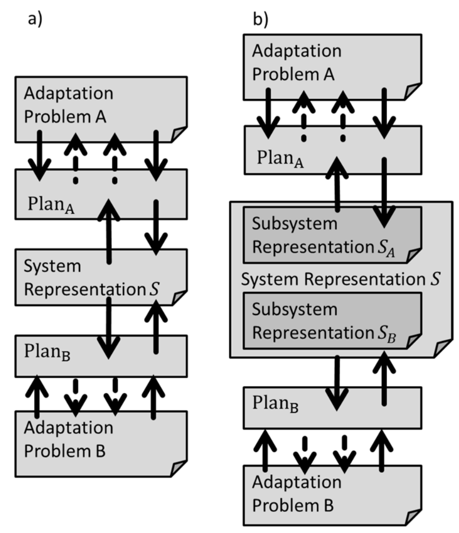

Figure 8: Illustration of a concurrent hybrid policy.

5.5 Hierarchical Combination

In a hierarchical combination of adaptation policies the adaptive system of one policy is part of the information used in the other policy. This means, the purpose of the second policy is not to adapt the same adaptive system but to change the way in which the first policy adapts the adaptive system.

This is illustrated in Fig. 9. Plan phase $Plan_B$ targets an adaptive system B that contains the adaptation problem of $Plan_A$. The purpose of this combination is to assure or optimize properties of the adaptation mechanism itself, like its calculation time or result quality. For this purpose the category B used by $Plan_B$ will often use additional information about the behaviour of $Plan_A$ that can be monitored and used as basis for decisions.

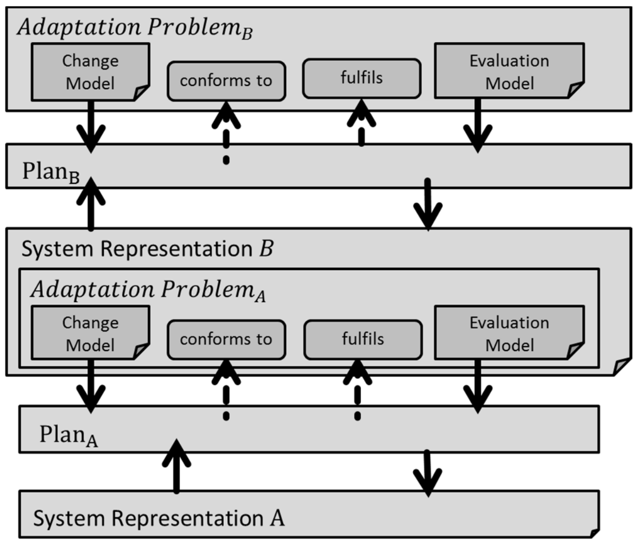

Figure 9: Illustration of a hierarchical hybrid policy.

In our model the hierarchical combination is represented by two separate plan phases, one of which targets information used in the other one. This information may not always be the adaptation problem. It may be only part of the adaptation problem (e.g., only the evaluation or change model) or other information used in $Plan_A$. For example, if $Plan_A$ contains an optimization approach, such as simulated annealing, the second policy could tweak parameters such as acceptance probability.

5.6 Discussion

In this section we present a classification of four types of combination for adaptation policies and a model that can be used to represent how these combinations are implemented. If two policies are combined with a single plan phase (i.e., in a single MAPE-K loop) they can be *integrated* or *coordinated*, based on whether the plan phases from both policies can still be distinguished. In the case of multiple plan phases they can be *concurrent*, if their adaptive systems coincide or are parts of the same overall system, or *hierarchical*, if the adaptive system of one is part of the plan phase or adaptation problem of the other. If neither case applies we say the policies are unrelated.

Since the concurrent and hierarchical case concern multiple MAPE-K loops similar concepts can be found in frameworks that model the interaction of MAPE-K loops. In EUREMA [21] the concurrent combination is called independent or coordinated execution of feedback loops while the hierarchical combination is called layered execution of feedback loops.

The classification presented in this section is limited to the combination of two policies. However, the complexity of realistic approaches often requires the combination of more than two policies or a combination in more than one type. Since the combined policies do not need to be pure policies the four types of combination can be applied multiple times, thus nesting these combinations. For example, two policies could be *integrated* with each other and the resulting policy could be optimized via a third policy in *hierarchical* combination.

The classification could also be extended to more detailed subcategories. For *integrated* and *coordinated* combination, subcategories could be based on different combinations of adaptation problems or different implementations of the plan phase. The type of combined policy could also be taken into consideration. This could reveal whether there are special types of combination based on which policies are combined or which information is contained in the combined policies.

6 APPLICATION

In this section, we apply our model to DeltaIOT and L-DSPL to illustrate how these approaches are modelled and explain the differences we observed in Section 3. These frameworks are examples of *integrated* and *hierarchical* combination. After discussing them we give examples of the other two combinations.

DeltaIOT is a mix of a utility-based and goal-based policy. The main adaptation mechanism is goal-based, but uses utility functions as side goals. Since the adaptation problem is solved in one plan phase this is an example of an *integrated* adaptation policy. The model of DeltaIOT is given in Fig. 10.

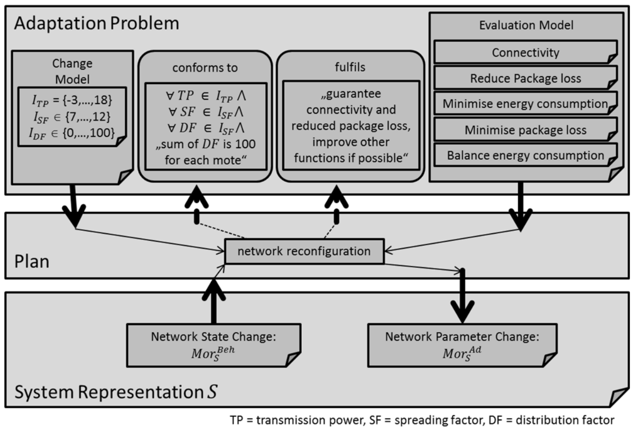
Adaptation Problem
Change Model
I_{TP} = {-3,…,18}
I_{SF} ∈ {7,…,12}
I_{DF} ∈ {0,…,100}
conforms to
∀ TP ∈ I_{TP} ∧
∀ SF ∈ I_{SF} ∧
∀ DF ∈ I_{SF} ∧
„sum of DF is 100 for each mote"
fulfils
„guarantee connectivity and reduced package loss, improve other functions if possible"
Evaluation Model
Connectivity
Reduce Package loss
Minimise energy consumption
Minimise package loss
Balance energy consumption
Plan
network reconfiguration
Network State Change: Mor_S^{Beh}
Network Parameter Change: Mor_S^{Ad}
System Representation S
TP = transmission power, SF = spreading factor, DF = distribution factor

Figure 10: Model of DeltaIOT

The adaptive system representation of DeltaIOT consists of network parameters for each connection and information about the network state.

The *change model* prescribes intervals for transmission power, spreading factor and distribution factor. A network configuration *conforms to* this change model if the factors of each connection are within those intervals and if the sum of distribution factors for all connections from one node is 100 (Iftikhar et al. state that the sum is usually 100 [10]). The *evaluation model* contains Boolean goals (maintain connectivity and reduce package loss) and utility functions (minimise energy consumption, minimise package loss and balance energy consumption). The Boolean goals are required by relation *fulfils*. In addition, this relation requires the utility functions to be improved if possible. This is a reflection of the fact that the approach optimizes these functions gradually.

The adaptation mechanism in DeltaIOT changes the network parameters gradually. It does so by improving the utility functions step-wise until no more improvement can be made.

L-DSPL is a combination of a rule-based adaptation policy (DSPL) and goal-based adaptation policy (learning). Both have separate planning phases and learning optimizes the rules, which are part of the DSPL approach. This is a *hierarchical* combination of those policies. The model of L-DSPL is given in Fig. 11.

The lower part of the figure represents the DSPL approach. The category representing the adaptive system contains an instance of the feature model, which represents the current state of the software system, and a context model, which represents the state of the monitored context information. The adaptation problem contains a set of adaptation rules (*change model*). In the DSPL exactly one rule is applied at the same time (*conforms to*). As usual in rule-based approaches there is no explicit notion of goal as the goal is encoded in the rules. Thus, the *evaluation model* is empty and always *fulfilled*. The plan phase selects a rule and then switches the running system to adhere to this rule.

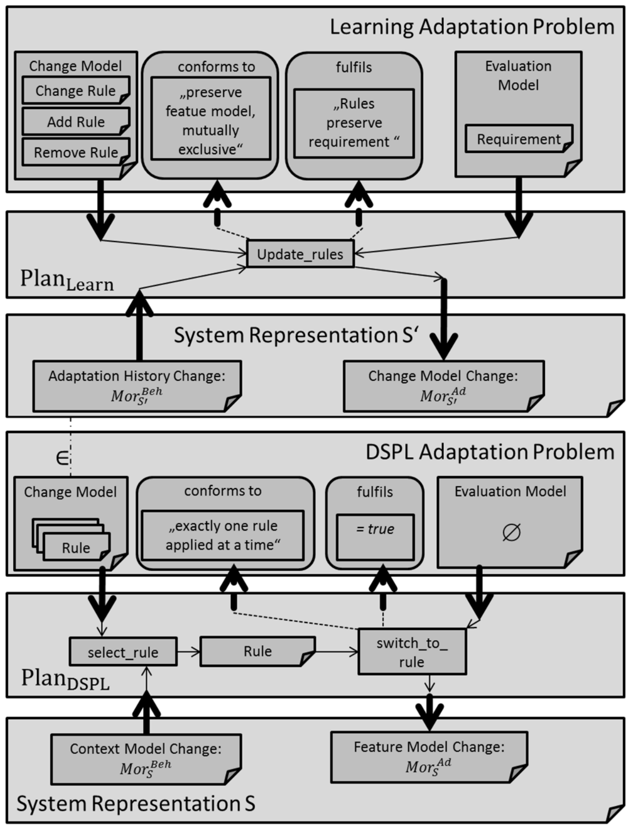
Learning Adaptation Problem
Change Model
Change Rule
Add Rule
Remove Rule
conforms to
„preserve feature model, mutually exclusive"
fulfils
„Rules preserve requirement "
Evaluation Model
Requirement
Update_rules
Plan_Learn
System Representation S'
Adaptation History Change: $Mor_{S'}^{Beh}$
Change Model Change: $Mor_{S'}^{Ad}$
∈
DSPL Adaptation Problem
Change Model
Rule
conforms to
„exactly one rule applied at a time"
fulfils
= true
Evaluation Model
∅
select_rule
Rule
switch_to_rule
Plan_DSPL
Context Model Change: Mor_{S}^{Beh}
Feature Model Change: Mor_{S}^{Ad}
System Representation S

Figure 11: **Model of L-DSPL**

The upper part of the figure represents the learning component. The adaptive system of this plan phase contains the change model of the DSPL part. It also contains information about the adaptation history, i.e., in which context states the rules have worked and failed. The adaptation problem is goal-based. The *change model* contains actions to change, add or remove rules from the DSPL change model. A feature model *conforms to* this change model if all rules adhere to the feature model and have mutually exclusive conditions. The *evaluation model* contains a Boolean requirement, that needs to be preserved (*fulfils*).

Using these models we are able to identify the difference in combining policies described in the end of Section 3. While both examples combine two adaptation policies, they do so using different types of combination. DeltaIOT uses an *integrated* combination in which the adaptation problem uses information from both policies. L-DSPL uses a *hierarchical* combination in which one policy adapts the other one.

Examples for the other two types of combination can also be found. The hybrid planning approach by Pandey et al. is an example for a *coordinated* combination [14]. This approach solves an integrated planning problem (goal-based problem with utility function) by using multiple planning algorithms and selecting among the results based on time- and quality constraints.

An example for a *concurrent* approach is the self-adaptation approach by Vogel and Giese [19] in which a model of an adaptive system is synchronized with multiple views on that model which reflect only the information relevant to a specific aspect (e.g., component failure or performance). Each partial model has its own adaptation mechanism.

7 DISCUSSION AND FUTURE WORK

In this paper we classify four ways to combine adaptation policies into hybrid policies: *integrated*, *concurrent*, *coordinated* and *hierarchical*. The classification is done based on which adaptive system they target, whether they solve the same adaptation problem and whether their plan phases are integrated. To represent these combinations we use a reference model, representing the implementation of an adaptation policy in terms of functions, models and relations. The reference model uses category theory as formal foundation to represent the state space of an adaptive system.

The classification and the reference model can be used to structure the development of self-adaptive systems and compare different approaches based on how their adaptation mechanism is implemented. While the classification is still fairly course, the accompanying model enables to represent the implementation and combination of approaches in more detail and can be used for more fine-grained discussion and comparison of approaches.

The model and classification are the result of our state of the art analysis on SEAMS 2016 and 2017. While we believe the resulting sample set of approaches to be representative, it may be the case that the classification needs extension in future work to capture cases that haven't been encountered yet.

To evaluate completeness and extend our classification and model we aim to extend the structured literature analysis and model each identified approach with our model. As a result of this activity we can further refine classification and model by considering repeated patterns in the resulting models.

The model provided in this paper is an early version of a modelling language that aims to model plan phases to classify and compare them. Since we expect further extensions due to the above mentioned refinement activities, we left the model on an abstract level, avoiding a completely specified abstract syntax. In future work we intend to complete and provide this model to the community. The implementation will likely be a mapping to or extension of existing modelling languages. Candidates that have been considered and excluded for this paper where FORMS [24], which provides more of an architectural view than the input-output-based process view in our model, and EUREMA [21], which is not able to represent relations. It is likely that the final implementation will constitute an extension of or mapping to such a modelling language.

REFERENCES

[1] Balasubramanian, S., Desmarais, R., Müller, H. A., Stege, U., Venkatesh, S.: Characterizing problems for realizing policies in self-adaptive and self-managing systems. In: Proceedings of the 6th international Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS2011), 70–79. ACM Press (2011)

[2] Baldan, P., Corradini, A., Heindel, T., König, B., Sobocinski, P.: Unfolding grammars in adhesive categories. In: Algebra and Coalgebra in Computer Science, Volume 5728 of Lecture Notes in Computer Science, 350–366.

Springer-Verlag (2009)

[3] Born, K., Lambers, L., Strüber, D, Taentzer, G.: Granularity of Conflicts and Dependencies in Graph Transformation Systems. In: Proceedings of the 10th International Conference on Graph Transformation (ICGT2017), 125–141. Springer-Verlag (2017)

[4] Ehrig, H., Golas, U., Hermann, F.: Categorical Frameworks for Graph Transformation and HLR Systems based on the DPO Approach. Bulletin of the European Association for Theoretical Computer Science 102, 111–121. (2010)

[5] IBM: Autonomic computing - the 8 elements (2001), http://researchweb.watson.ibm.com/autonomic/overview/elements.html

[6] Herrlich, H., Strecker, G.: Category theory: an introduction. Allyn and Bacon series in advanced mathematics. Allyn and Bacon (1973)

[7] Hinchey, M., Park, S., Schmid, K.: Building dynamic software product lines. In: IEEE Computer, 45(10):22–26. IEEE Computer Society (2012)

[8] Hussein, M., Han, J., Colman, A.: Specifying and verifying the context-aware adaptive behaviour of software systems. Tech. Rep. C3-516-03, Swinburne University of Technology, Faculty of Information and Communication Technologies (FICT) (December 2010)

[9] Hussein, M., Han, J., Colman, A.: Context-aware adaptive software systems: A system-context relationships oriented survey. Tech. Rep. C3-516-01, Swinburne University of Technology, Faculty of Information and Communication Technologies (FICT) (2010)

[10] Iftikhar, M. U., Ramachandrany, G. S., Bollansée, P., Weyns, D., Hughes, D.: DeltaIoT: A Self-Adaptive Internet of Things Exemplar. In: Proceedings of the 12th international Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS2017), 76–82. IEEE Press (2017)

[11] Kephart, J. O., Chess, D. M.: The vision of autonomic computing. In: IEEE Computer 36(1), 41–50, IEEE Computer Society (2003)

[12] Kephart, J. O., Walsh, W. E.: An articial intelligence perspective on autonomic computing policies. In: POLICY. 3–12. IEEE Computer Society (2004)

[13] Object Management Group: Business Process Model and Notation (BPMN), Version 2.0, (Technical report, Object Management Group)

[14] Pandey, A., Ruchkin, I., Schmerl, B., Cámara, J.: Towards a Formal Framework for Hybrid Planning in Self-Adaptation. In: Proceedings of the 12th international Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS2017), 109–115. IEEE Press (2017)

[15] Russel S., Norvig P.: Artificial Intelligence: A Modern Approach, 2nd Edition, Prentice Hall (2002)

[16] Sharifloo, M. A., Metzger, A., Quinton, C., Baresi, L., Pohl, K.: Learning and evolution in dynamic software product lines. In: Proceedings of the 11th international Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS2016), 156–164. IEEE Press (2016)

[17] Trollmann, F., Albayrak, S.: Extending Model to Model Transformation Results from Triple Graph Grammars to Multiple Models. In: Proceedings of the 8th International Conference on Model Transformation (ICMT2015), 214–229. (2015)

[18] Villegas, N. M., Tamura, G., Müller, H. A., Duchien, L., Casallas, R.: DYNAMICO: A Reference Model for Governing Control Objectives and Context Relevance in Self-Adaptive Software Systems. In: Software Engineering for Self-Adaptive Systems II, Lecture Notes in Computer Science, vol. 7475, 265–293. Springer-Verlag (2013)

[19] Vogel, T., Giese, H.: Adaptation and Abstract Runtime Models. In: Proceedings of the Workshop on Software Engineering for Adaptive and Self-Managing Systems, 39–48. ACM Press (2010),

[20] Vogel T., Seibel, A., Giese, H.: The Role of Models and Megamodels at Runtime. In: Models in Software Engineering. MODELS 2010. Lecture Notes in Computer Science, vol. 6627, 224–238. Springer (2011),

[21] Vogel, T., Giese, H.: Model-Driven Engineering of Self-Adaptive Software with EUREMA. ACM Transactions on Autonomous Adaptive Systems 8(4), 18:1-18:33 (Jan 2014)

[22] Vromant, P., Weyns, D., Malek, S., Andersson, J.: On interacting control loops in self-adaptive systems. In: Proceedings of the 6th international Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS2011), 202–207. ACM Press (2011)

[23] Weyns, D., Malek, S., Andersson, J.: On decentralized self-adaptation: Lessons from the trenches and challenges for the future. In: Proceedings of the Workshop on Software Engineering for Adaptive and Self-Managing Systems. 84–93. ACM Press (2010)

[24] Weyns, D., Malek, S., Andersson, J.: Forms: Unifying reference model for formal specification of distributed self-adaptive systems. ACM Transactions on Autonomous Adaptive Systems 7(1), 8:1–8:61, ACM (May 2012)

[25] Zhang, J., Cheng, B. H. C., Goldsby, H.: Amoeba-rt: Run-time verification of adaptive software. In: Models in Software Engineering, Workshops and Symposia at MoDELS 2007, Reports and Revised Selected Papers. Lecture Notes in Computer Science, vol. 5002 212–224. Springer-Verlag (2007)