View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Open Repository and Bibliography - Luxembourg

A Temporal Model for Interactive Diagnosis of Adaptive Systems

Ludovic Mouline*†, Amine Benelallam†, François Fouquet‡, Johann Bourcier†, Olivier Barais†
*University of Luxembourg, firstname.lastname@uni.lu
†Univ Rennes, Inria, CNRS, IRISA, firstname.lastname@inria.fr
‡DataThings S.A.R.L Luxembourg, francois.fouquet@datathings.com

***Abstract*—The evolving complexity of adaptive systems impairs our ability to deliver anomaly-free solutions. Fixing these systems require a deep understanding on the reasons behind decisions which led to faulty or suboptimal system states. Developers thus need diagnosis support that trace system states to the previous circumstances –targeted requirements, input context– that had resulted in these decisions. However, the lack of efficient temporal representation limits the tracing ability of current approaches. To tackle this problem, we describe a novel temporal data model to represent, store and query decisions as well as their relationship with the knowledge (context, requirements, and actions). We validate our approach through a use case based on the smart grid at Luxembourg.**

***Keywords*-adaptive systems, traceability, diagnosis, model-driven engineering**

I. Introduction

Adaptive systems have the capabilities of adjusting their behavior to dynamic changes encountered in their environments. To do so, they make adaptation decisions, in the form of actions, based on high-level policies. These adaptations are often relevant to achieving a goal or an objective while satisfying a set of constraints [1]. One of the successful approaches used to achieve this adaptability is the closed control loop paradigm, in particular, the MAPE-K loop [2]. It consists of monitoring the well-behaving of the system with regards to a set of requirements, either functional or non-functional. Once the system deviates from its normal behavior, the control loop selects and executes a set of actions (adaptation tactic) to remediate the system behavior.

Faced with growingly complex and large-scale software systems (e.g. smart grid systems), the presence of residual defects becomes unavoidable [3]. As there might be many probable causes behind an abnormal behavior, developers usually perform a set of diagnosis routines to narrow down the scope or origin of the failure. This is achieved through the investigation of requirements satisfaction and the decisions that led to this particular system state, as well as their timing [4]. In this perspective, developers will set up a set of systematic questions to understand why and how the system is behaving in such a way.

Bencomo *et al.,* [4] argue that comprehensive explanation about the system behavior contributes drastically to the quality of the diagnosis, and eases the task of troubleshooting the system behavior. To enable this, we believe that adaptive software systems should be equipped with traceability management facilities to link the decisions made to their **(i) circumstances, that is to say, the history of the system states and the targeted requirements, and (ii) the performed actions with their impact(s) on the system**. In particular, an **adaptive system should keep a trace of the relevant historical events**. Additionally, it should be able to **trace the goals intended to be achieved by the system to the adaptations and the decisions that have been made, and vice versa**. Finally, in order to enable developers to interact with the system in a clear and understandable way, appropriate abstraction to **enable the navigation of the traces and their history should also be provided**.

Existing approaches [5]–[10] are accompanied by built-in monitoring rules and do not allow to interact with the underlying system in a simple way. Moreover, they do not keep track of historical changes as well as causal relationships linking requirements to their corresponding adaptations. Only flat execution logs are stored. In this paper, we propose a framework to structure and store the state and behavior of a running adaptive system, together with a high-level API to efficiently perform diagnosis routines. Our framework relies on a temporal model-based solution that efficiently abstracts decisions and their corresponding circumstances.

The rest of the paper is structured as follows. We first describe a guidance example in Section II, based on the smart grid system at Luxembourg. In Section III, we summarize core concepts manipulated in adaptation processes and their characteristics. Later, we describe the proposed data model in Section IV. In Section V, we demonstrate the applicability of our approach by applying it to the smart grid example. Before concluding the paper in Section VII, we introduce some related work in Section VI.

II. Guidance example: Smart Grid

The National Institute of Standards and Technology defines smart grids as “a modernized grid that enables [...] **control capabilities** that will lead to a collection of **new functionalities and applications**”. These capabilities can be implemented using approaches developed by the community of adaptive systems.

Hartmann *et al.,* [11] describe the smart grid at Luxembourg as a hierarchical system composed of three elements: central system (CS), data concentrator (DC), and smart meters (SM). SM regularly measure resource consumption (*e.g.,* electricity) and report them to the CS through the DC. A smart meter

can also modify the maximal consumption or even cut off the resource. In addition to storing the consumption data, DC autonomously manage SM according to the configuration sent by the CS.

Among the different goals of DC, they have to minimize the number of overloads on the network. In the remaining part of the paper, we refer to this goal as the "*minimizing overload*" policy. They have two action points: either on the production side or the consumption side. They can reduce or increase the production by (dis)connecting production unit or the consumption by modifying the maximum permitted consumption. We called these actions: *reduce production*, *increase production*, *reduce amps limit* and *increase amps limit*. However, as all adaptive systems, smart grids are prone to failures [12]. Using our approach, an engineer could diagnose the system, and determine the adaptation process responsible for this failure. For instance, considering some reports about regular power cuts during the last couple of days, in a particular area, a stakeholder may want to interrogate the system and determine what past decision(s) have led to this suboptimal state. More concretely, he will ask: did the system make any decisions that could have impacted the customer consumption? If so, what goal(s) the system was trying to reach and what were the values used at the time the decision(s) was(were) made?

III. Background: Knowledge

In this section, we abstract common concepts implied in an adaptation process, that is, context, decisions, and their circumstances. We refer to these concepts as the knowledge.

A. General concepts of adaptation process

IBM defines adaptive systems as "a computing environment with the ability to manage itself and **dynamically adapt** to change in accordance with **business policies and objectives**. [These systems] can perform such activities based on **situations they observe or sense in the IT environment** [...]" [13].

Based on this definition, we can identify three principal concepts involved in adaptation processes. The first concept is *actions*. They are executed in order to perform a dynamic adaptation through actuators. The second concept is **business policies and objectives**, which is also referred to as the **system requirements** in the domain of (self-)adaptive systems. The last concept is the observed or sensed **situation**, also known as the **context**. The following subsections provide more details about these concepts.

B. Context

While several works [14]–[17] have been proposed to specify the defining characteristics of context data, in this section, we list some of these characteristics in order to justify the design choices of our Knowledge meta-model (cf. Section IV).

a) Volatility: Data can be either **static** or **dynamic**. Static data, also called frozen, are data that will not be modified, over time, after their creation [14]. For example, the location of a machine, the first name or birth date of a user can be identified as static data. Dynamic data are, instead, subject to modification over time.

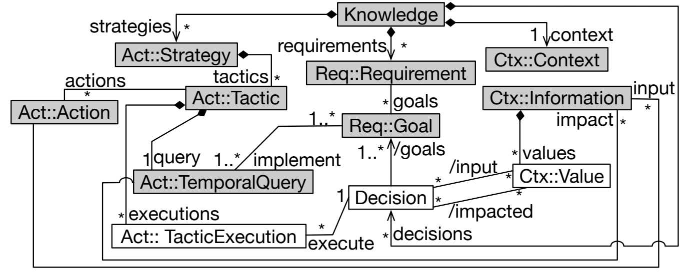

Fig. 1. Excerpt of the knowledge metamodel

b) Temporality: In dynamic data, sometimes we may be interested not only in storing the latest value, but also the previous ones [14], [15], in order to analyze the data evolution trend for example. We refer to these data as **historical** data.

c) Uncertainty: One of the recurrent problems facing context-aware applications is the data uncertainty [16]. Uncertain data are not likely to represent the reality. They contain a noise that makes it deviate from its original value.

d) Source: According to the literature, data sources are grouped into two main categories, either sensed (measured) data or computed (derived) data [14], [17]. We refine this with an additional category called profiled. Profiled data may be set either by a user (**profiled by a human**) or by an external system (**profiled by an external**).

e) Connection: Context data entities are usually linked using three kinds of connections: conceptual, computational, and consistency [16]. The conceptual connection relates to (direct) relationships between entities in the real world (e.g. smart meter and concentrator). The computational connection is set up when the state of an entity can be linked to another one by a computation process (derived, predicted). Finally, the consistency connection relates entities that should have consistent values.

C. Requirement

Adaptation processes aim at modifying the system state to reach an optimal one. All along this process, the system should respect the **system requirements** established ahead. Although in the literature, requirements are categorized as functional or non-functional, in this paper we use a more elaborate taxonomy introduced by Glinz [18], which classifies requirements in four categories: functional, performance, specific quality, and constraint.

D. Action

In adaptive systems, actions are defined as a process that, given the context and requirements as input, adjusts the system behavior. These modification may create new data that correspond to an output context. In the remainder of this paper, we refer to output context as impacted context, or simply impact(s). Whereas requirements are used to add preconditions to the actions, context information is used to drive the modifications.

IV. Modeling the Knowledge

In order to simplify the diagnosis of adaptive systems, in this paper, we propose a novel metamodel that combines, what we call, design elements and runtime elements. Design elements abstract the different elements involved in knowledge information to assist the specification of the adaptation process. Runtime elements instead, represent the data collected by the adaptation process during its execution. In order to maintain the consistency between previous design elements and newly created ones, instances of design elements (*e.g.,* actions) can be either added or removed. Modifying these elements would consist in removing existing elements and creating new ones. Design time elements are depicted in gray in the Figures 1–4. Note that, in this paper, we do not address how runtime information is collected.

For the sake of modularity, we split our metamodel into four packages. First, we describe the Knowledge (core) package. Then, we introduce in more details the other three packages used by the knowledge package: Context, Requirement, and Action. We assume that all the classes in the different packages extend a *TimedElement* class. This class contains three methods: *startTime*, *endTime* and *modificationsTime*. The first two methods allow accessing the validity interval of an object. The last method resolves all the timestamps at which an element has been modified: its history.

A. Knowledge metamodel

In order to enable interactive diagnosis of adaptive systems, we claim that traceability links between the decisions made and their circumstances should be organized in a well-structured representation. In what follows, we introduce the knowledge metamodel. It describes how decisions are linked to the goals and the context (input and impact). Figure 1 depicts this metamodel. We use "*Package::Class*" notation to refer to the provenance of a class, in case it belongs to another package.

Knowledge is composed of a *context*, a set of *requirements*, a set of *strategies*, and a set of *decisions*. A decision can be seen as the output of the Analyze and Plan steps in the MAPE-k loop. Decisions comprise target *goals* and trigger the execution of one *tactic* or more. A decision has an *input* context and an *impacted* context. The context impacted by a decision (*Decision.impacted*) is a derived relationship computed by aggregating the impacts of all actions belonging to a decision (see Fig. 4). Likewise, the *input* relationship is derived and can be computed similarly. In the smart grid example, a decision can be formulated (in plain English) as follows: since the district D is almost overloaded (*input context*), we reduce the amps limit of greedy consumers using the "*reduce amps limit*" *action* in order to reduce the load on the cable of the district (*impact*) and satisfy the "*minimizing overload*" policy (*requirement*).

As all the elements inherit from the *TimedElement*, we can capture the time at which a given decision and its subsequent actions were executed, and when their impact materialized, *i.e.,* measured. Thanks to this metamodel representation, we can apprehend the possible causes behind malicious behavior by navigating from the context values to the decisions that have impacted its value (*Information.values.impact*), then the goals it was trying to reach (*Decision.goals*). In Section V, we present an example of interactive diagnosis queries applied to the smart grid use case.

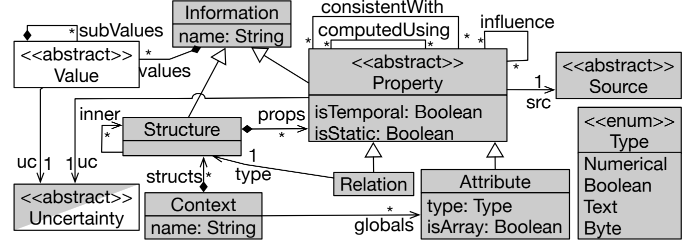

Fig. 2. Excerpt of the context metamodel

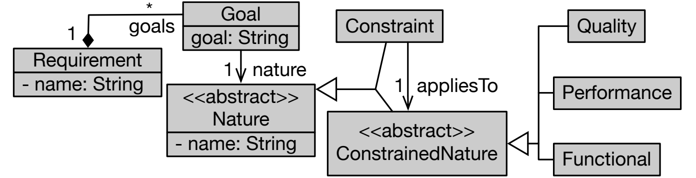

Fig. 3. Requirement metamodel

B. Context metamodel

Context models structure context information acquired at runtime. For example, in a smart-grid system, the context model would contain information about smart-grid users (address, names, etc.) resource consumption, etc.

An excerpt of the context model is depicted in Figure 2. We propose to represent the context as a set of structures (*Context.structs*) and global attributes (*Context.globals*). A structure can be viewed as a C-structure with a set of properties (*Property*): attributes (*Attribute*) or relationships (*Relation*). A structure may contain other nested structures (Structure.inner). Structures and properties have values. The connection feature described in Section III-B is represented thanks to three recursive relationships on the Property class: *consistentWith*, *computedUsing* and *influence*. Additionally, each property has a source (*Source*) and an uncertainty (*Uncertainty*). It is up to the stakeholder to extend data with the appropriate source: measured, computed, provided by a user, or by another system (*e.g.,* weather information coming from a public API). Similarly, the uncertainty class can be extended to represent the different kinds of uncertainties. Finally, a property can be either historic or static.

C. Requirement metamodel

As different solutions to model system requirements exist (*e.g.,* KAOS [19] or i* [20]), in this metamodel, we abstract their shared concepts. Our requirement model, depicted in Figure 3, represents the *requirement* as a set of *goals*. Each goal has a *nature* and a textual specification. The nature

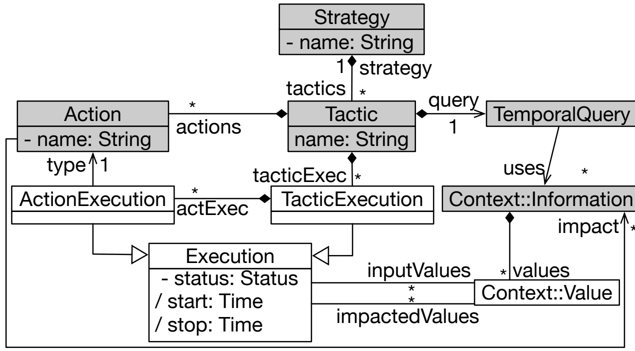

Fig. 4. Excerpt of the action metamodel

of the goals adheres to the four categories of requirements presented in Section III-C. We may use one of the existing requirements modeling languages (*e.g.,* RELAX) to define the semantics of the requirements. Since the requirement model is composed solely of design elements, we may rely on static analysis techniques to infer the requirement model from existing specifications. The work of Egyed [21] is one solution among others. This work is out of the scope of the paper and envisaged for future work.

In our guidance example, the requirement model may contain a **balanced resource distribution** requirement. It can be split into different goals: (i) *minimizing overloads*, (ii) *minimizing lacks of production*, (iii) *minimizing production loss*.

D. Action metamodel

Similar to the requirements metamodel, the actions metamodel also abstracts main concepts shared among existing solutions to describe adaptation processes and how they are linked to the context. Figure 4 depicts an excerpt of the action metamodel. We define a strategy as a set of tactics (*Strategy*). A tactic contains a set of actions (*Action*). A tactic is executed under a precondition represented as a temporal query (*TemporalQuery*) and uses different data from the context as input. In future work, we will investigate the use of preconditions to schedule the executions order of the actions, similarly to existing formalisms such as Stitch [22]. Finally, actions have impacts on certain properties, represented by the *impacted* reference.

The different executions are represented thanks to the *Execution* class. Each execution has a status to track its progress and links to the impacted context values(*Execution.impactedValues*). Similarly, input values are represented thanks to the *Execution.inputValues* relationship. An execution has *start* and *end* time. Not to confuse with the *startTime* and *endTime* of the *TimedElement* class. Whilst the former corresponds to the time range in which a value is valid, the *start* and *stop* time in the class execution correspond to the time range in which an action or a tactic was being executed.

Similarly to requirement models, it is possible to automatically infer design elements of action models by statically analyzing actions specification. Since acquiring information about tactics and actions executions happens at runtime, one way to achieve this is by intercepting calls to actions executions and updating the appropriate action model elements accordingly. This is out of the scope of this paper and planned for future work.

In the next section, we describe how our approach can be applied to the smart grid system. We show how one can implement model manipulation operations involved in diagnosis algorithms. Prior to this, we briefly introduce the implementation of our approach on top of an existing framework to store temporal models.

V. Application on the smart grid example

To validate our approach, we implemented a prototype publicly available online [1]. This implementation leverages the GreyCat framework[2], more precisely the modeling plugin, which allows the management and persistence of temporal models.

In what follows, we explain how a stakeholder, Morgan, can apply our approach to a smart grid system in order to, first, abstract adaptive system concepts, then, structure runtime data, and finally, query the model for diagnosis purpose. The corresponding object model is depicted in Figure 5. Due to space limitation, we only present an excerpt of the knowledge model. An elaborate version is accessible in the tool repository.

*a) **Abstracting the adaptive system**:* At design time (t_d), either manually or using an automatic process, Morgan abstracts the different tactics and actions available in the adaptation process. Morgan would like to model one adaptation tactic called "*reduce amps limit*". It is composed of three actions: sending a request to the smart meter (*askReduce*), checking if the new limit corresponds to the desired one (*checkNewLimit*), and notifying the user by e-mail (*notifyUser*). Morgan assumes that the *askReduce* action impacts consumption data (*csmpt*). This tactic is triggered upon a query (*tempQ*) that uses meter (*mt*), consumption (*csmpt*) and customer (*cust*) data. The query implements the "*minimizing overload*" goal: the system shall minimize the number of overloaded cables. Figure 5 depicts a flattened version of the temporal model representing these elements. The tag at upper-left corner of every object illustrates the creation timestamp. All the elements created at this stage are tagged with t_d.

*b) **Adding runtime information**:* The adaptation process checks if the current system state fulfills the requirements by analyzing the context. To perform this, it executes the different temporal queries, including *tempQ*. For some reasons, the *tempQ* reveals that the current context does not respect the "*minimizing overload*" goal. To adapt the smart grid system, the adaptation process decides to start the execution of the previously described tactic (*exec1*) at t_s. As a result, a decision element is added to the model along with a relationship to the unsatisfied goal. In addition, this decision entails the planning of a tactic execution, manifested in the creation of the element *exec1* and its subsequent actions (*notifyU*, *chckLmt*, and *askRdc*). At t_s, all the actions execution have an IDLE

[1] https://github.com/lmouline/LDAS
[2] https://github.com/datathings/greycat

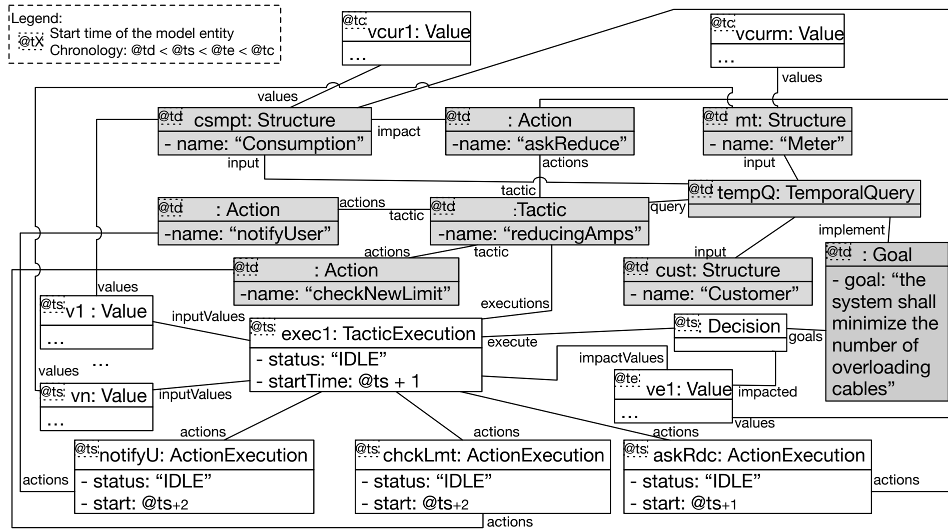

Fig. 5. Excerpt of the knowledge object model related to our smart grid example

status and an expected start time. All the elements created at this stage are tagged with the t_s timestamp in Figure 5.

At t_{s+1}, the planned tactic starts being executed by running *askRdc*. The status of this action execution turns from *IDLE* to *RUNNING*. Later, at t_{s+2}, the execution of *askRdc* finishes with a *SUCCEED* status and triggers the action execution *notifyU* and *chckLmt* in parallel. The status of *askRdc* changes to *SUCCEED* while the status of *notifyU* and *chckLmt* turns to *RUNNING*. The first action execution successfully ends at t_{s+3} while the second ends at t_{s+4}. As all action executions terminate with a *SUCCEED* status at t_{s+4}, accordingly, the final status of the tactic is set *SUCCEED* and the *stop* attribute value is set to t_e.

c) ***Interactive diagnosis query****:* After receiving incident reports concerning regular power cuts, and based on the aforementioned knowledge model, Morgan would be able to query the system's states and investigate why such incidents have occurred. As described in Section II, she/he will interactively diagnose the system by interrogating the context, the decisions made, and their circumstances.

```
// extracting the decisions
Decision[] impactedBy(Value v) {
  Decision[] respD
  for( Time t: v.modificationTimes() ):
    if (t >= v.startTime() - 2 day)
      Value resV = resolve(v,t)
  respD.addAll(from(resV).navigate(Value.impacted))
  return respD
  }
// extracting the circumstances of the made decisions
Tuple<Value[], Goal[]> getCircumstance(Decision d) {
  Value[] resValues = from(d).navigate(Decision.input)
  Goal[] resGoals = from(d).navigate(Decision.goals)
  return Tuple<>(resValues, resGoals)
  }
```

Listing 1. Get the goals used by the adaptation process from executed actions

The first function, depicted in Listing 1, allows to navigate from the currently measured values (*vcur1*) to the decision(s) made. The for-loop and the if-condition are responsible for resolving the measured data for the past two days. After extracting the decisions leading to power cuts, Morgan carries on with the diagnosis by accessing the circumstances of this decision. The code to perform this task is depicted in Listing 1, the second function (getCircumstances). Note that the relationship *Decision.input* is the aggregation of *Decision.excecute.inputValues*.

VI. Related work

Many research efforts focusing on analyzing and monitoring the behavior of self-adaptive software systems have been carried out. In particular, some approaches suggest the use of formal methods and languages for detecting abnormal behavior and deciding reconciliation scenarios. Arcaini *et al.,* [23] proposed an approach for formal modeling, validation, and verification of distributed self-adaptive systems by relying on on the concept of multi-agent Abstract State Machines. Another approach [24] specifies goals, requirement conditions, and their satisfaction rate using probabilistic linear temporal logic (PLTL). These assertions are further used to monitor the system at runtime. When the satisfaction rate obtained for high-level goals is weak, a cost-benefit trade-off analysis is performed to select goals maximizing cost constraints. FORMS [25] is a formal reference model for the specification of distributed adaptive systems. It consists of a set of primitives and the relationships of formally specified modeling elements that correspond to the key concerns in the design of self-adaptive systems and a set of relationships that guide their composition. On the contrary to our approach, these solutions perform only at runtime. They are not able to reconstruct execution traces at postmortem. Furthermore, given the temporal properties of the underlying logic, these approaches are not able to capture physical and continuous time. Our approach can be seen as complementary to these approaches since it exposes similar core concepts.

Another family of approaches is based on model-driven techniques, namely the models@runtime paradigm. Ehlers *et*

al., [7] proposed a rule-based approach for performance anomaly localization for adaptive systems. The monitoring rules, specified in OCL (Object Constraint Language) reflect the specified goals at the component level. They refer to performance attributes, in particular, responsiveness anomaly scores, that change their values during runtime. Compared to our approach, this approach is applied only to performance anomalies. Henrich *et al.,* [6] suggested a model-based approach for the specification of causal relations linking the measured data and a component-based runtime prediction model. The authors opted for flat structural representation to store these links. Furthermore, they used aggregation functions in the form of formal rules to gather this information. As opposed to our approach, the proposed solution does not handle tracing error-prone records back to the corresponding circumstances. Moreover, it does not capture the history of goals execution, which improves detecting performance anomalies and determining the failure timing.

To the best of our knowledge, one close work of our work is the approach proposed by Bencomo *et al.,* [4]. They propose to use requirements monitoring to enable the explanation feature on adaptive systems. By extending a goal model with a claim-refinement model, they define the characteristics of the systems that will be monitored. According to the state of these elements, they can trace back to a goal and explain why the system has adapted his behavior. Their approach use requirement monitoring techniques whereas we provide a data model to structure runtime information and link them with design information, for example, goals and requirements.

VII. Conclusion & future work

Adaptive systems are prone to faults given their evolving complexity. To enable interactive diagnosis over these systems, we proposed a temporal data model to abstract and store knowledge elements. We also provided a high-level API to specify and perform diagnosis algorithms. Thanks to this structure, a stakeholder can abstract and store decisions made by the adaptation process and link them to their circumstances –targeted requirements and used context– as well as their impacts. To demonstrate the applicability of our approach, we applied it to a smart-grid case study. In future work, we plan to evaluate the performance of our approach on a very large data-set coming from the Luxembourg smart-grid.

Moreover, throughout this work, we assumed that designers are able to link actions with their expected impacts at design time. However, this is not always true. Some impacts cannot be known in advance. In this perspective, in addition to the future plans already mentioned throughout the paper, we will investigate techniques to identify unknown impacts on the context model, for instance, by studying the use of machine learning techniques.

References

[1] F. D. Macías-Escrivá, R. Haber, R. Del Toro, and V. Hernandez, "Self-adaptive systems: A survey of current approaches, research challenges and applications," *Expert Systs. with Apps.*, vol. 40, no. 18, 2013.

[2] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, 2003.

[3] D. M. Barbosa, R. G. D. M. Lima, P. H. M. Maia, and E. Costa, "Lotus@ runtime: a tool for runtime monitoring and verification of self-adaptive systems," in *SEAMS, 2017 IEEE/ACM 12th Int. Symp.* IEEE, 2017.

[4] N. Bencomo, K. Welsh, P. Sawyer, and J. Whittle, "Self-explanation in adaptive systems," in *ICECCS, 2012 17th Int. Conference*. IEEE, 2012.

[5] W. Hasselbring, R. Heinrich, R. Jung, A. Metzger, K. Pohl, R. Reussner, and E. Schmieders, "iobserve: Integrated observation and modeling techniques to support adaptation and evolution of software systems," Christian-Albrechts-Universität Kiel, Research Report, 2013.

[6] R. Heinrich, E. Schmieders, R. Jung, K. Rostami, A. Metzger, W. Hasselbring, R. Reussner, and K. Pohl, "Integrating run-time observations and design component models for cloud system analysis," in *Proc. of the 9th Workshop on Models@run.time*, vol. 1270. CEUR, 2014.

[7] J. Ehlers, A. van Hoorn, J. Waller, and W. Hasselbring, "Self-adaptive software system monitoring for performance anomaly localization," in *Proc. of the 8th ACM ICAC*. New York, NY, USA: ACM, 2011.

[8] D. F. Mendonça, R. Ali, and G. N. Rodrigues, "Modelling and analysing contextual failures for dependability requirements," in *SEAMS, 2014 IEEE/ACM 9th Int. Symp.* ACM, 2014.

[9] P. Casanova, D. Garlan, B. Schmerl, and R. Abreu, "Diagnosing unobserved components in self-adaptive systems," in *SEAMS, 2014 IEEE/ACM 9th Int. Symp.* ACM, 2014.

[10] M. U. Iftikhar and D. Weyns, "Activforms: Active formal models for self-adaptation," in *SEAMS, 2014 IEEE/ACM 9th Int. Symp.* ACM, 2014.

[11] T. Hartmann, F. Fouquet, J. Klein, Y. Le Traon, A. Pelov, L. Toutain, and T. Ropitault, "Generating realistic smart grid communication topologies based on real-data," in *SmartGridComm, 2014 IEEE Int. Conf.*, 2014.

[12] T. Hartmann, F. Fouquet, J. Klein, G. Nain, and Y. Le Traon, "Reactive security for smart grids using models@ run. time-based simulation and reasoning," in *Int. Workshop on Smart Grid Security*. Springer, 2014.

[13] "An architectural blueprint for autonomic computing," IBM, Tech. Rep., Jun. 2005.

[14] S. K. Chong, I. McCauley, S. W. Loke, and S. Krishnaswamy, "Context-aware sensors and data muffling," *Context awareness for self-managing systems (devices, applications and networks) proceeding*.

[15] T. Hartmann, F. Fouquet, G. Nain, B. Morin, J. Klein, and Y. Le Traon, "Reasoning at runtime using time-distorted contexts: A models@ run. time based approach," in *Proc. of the 26th Int. SEKE*. Knowledge Systems Institute Graduate School, USA, 2014.

[16] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, "A survey of context modelling and reasoning techniques," *Pervasive and Mobile Computing*, vol. 6, no. 2, 2010.

[17] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the internet of things: A survey," *IEEE communications surveys & tutorials*, vol. 16, no. 1, 2014.

[18] M. Glinz, "On non-functional requirements," in *RE Conference, 2007. 15th IEEE Int.* IEEE, 2007.

[19] A. Dardenne, A. Van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," *Science of computer programming*, vol. 20, no. 1-2, 1993.

[20] E. Yu, "Modelling strategic relationships for process reengineering," *Social Modeling for Requirements Engineering*, vol. 11, 2011.

[21] A. Egyed, "A scenario-driven approach to traceability," in *Proc. of the 23rd international conference on Software engineering*. IEEE Computer Society, 2001, pp. 123–132.

[22] S.-W. Cheng and D. Garlan, "Stitch: A language for architecture-based self-adaptation," *Journal of Systems and Software*, vol. 85, no. 12, 2012.

[23] P. Arcaini, E. Riccobene, and P. Scandurra, "Modeling and analyzing mape-k feedback loops for self-adaptation," in *SEAMS, 2015 IEEE/ACM 10th Int. Symp.* IEEE Press, 2015.

[24] A. Cailliau and A. van Lamsweerde, "Runtime monitoring and resolution of probabilistic obstacles to system goals," in *SEAMS, 2017 IEEE/ACM 12th Int. Symp.* IEEE, 2017.

[25] D. Weyns, S. Malek, and J. Andersson, "Forms: Unifying reference model for formal specification of distributed self-adaptive systems," *ACM TAAS*, vol. 7, no. 1, 2012.