Self-Adaptive Systems Planning with Model Checking using MAPE-K

Aristóteles Esteves Marçal da Silva[1], **Aline Maria Santos Andrade**[1], **Sandro Santos Andrade**[2]

[1]Computer Science Department – Federal University of Bahia (UFBA)
Salvador – Bahia – Brazil

[2]Computer Science Department – Federal Institute of Bahia
Salvador – Bahia – Brazil

totemarcal@gmail.com, aline@ufba.br, sandroandrade@ifba.edu.br

Abstract. *This paper presents a model checking-based approach to support the autonomous planning of adaptation actions in Self-Adaptive Systems, designed in consonance with the MAPE-K reference architecture. We evaluated our approach with a case-study aiming at verifying self-healing and self-organizing properties in a distributed and decentralized traffic monitoring system. Results show that our approach is able to generate adaptation plans satisfying the goals for all expected scenarios in such a case-study, providing a flexible formal framework where adaptation strategies and goals can be inserted/removed.*

Resumo. *Este artigo apresenta uma abordagem baseada em verificação de modelos para apoiar o planejamento autônomo de ações de adaptação em sistemas auto-adaptativos, projetados em consonância com a arquitetura de referência MAPE-K. Avaliamos nossa abordagem com um estudo de caso com o objetivo de verificar as propriedades de auto-recuperação e auto-organização em um sistema de monitoramento de tráfego distribuído e descentralizado. Os resultados mostram que nossa abordagem é capaz de gerar planos de adaptação que satisfazem as metas para todos os cenários esperados em um estudo de caso, fornecendo uma estrutura formal flexível onde estratégias e metas de adaptação podem ser inseridas / removidas.*

1. Introduction

Modern software systems are increasingly interacting with operating environments largely unpredictable and uncontrolled. In this scenario, solutions which adapt themselves in response to environmental and internal changes have been widely adopted in order to ensure the correct functioning of the system and keep delivering agreed service levels [Salehie and Tahvildari 2009a]. In dynamic systems, the operating environment and application's internal structure and behavior can change while the system is running [Russell and Norvig 2016].

A Self-Adaptive System (SAS) evaluates its own behavior and modifies itself whenever its primary purpose is not being effectively fulfilled, a better functionality or performance can be achieved [Laddaga 1997]. In the context of distributed systems, self-adaption increases reliability and contributes to endow the system with dependable capabilities. For example, hardware faults can be compensated by adapting the

system configuration and redistributing responsibilities amongst application components [Wildermann 2012].

Self-Adaptive System capable of performing self-diagnosis tasks when reacting to failures are usually named Self-Healing System (SHS). A system with many interacting elements with only partial knowledge about the global system state [Salehie and Tahvildari 2009b] – usually characterized by a decentralized control mechanism [Wildermann 2012] and an autonomous and decentralized cooperation [Andrade 2014] – is named a Self-Organizing System (SOS). The properties "self-healing," "self-protecting," "self-stabilizing," "self-organizing," and more are in conjunct are refered as "self-*"properties [Berns and Ghosh 2009].

The MAPE-K model is a reference architecture for SAS design and defines such systems as a group of four components: Monitor (M), Analyze (A), Plan (P) and Execute (E). Such components cooperate to implement a feedback loop with the help of a Knowledge Base (K) component [Kephart and Chess 2003].

Most SAS are critical and, therefore, require rigorous approaches in their development stages. The use of formal methods applied to Software Engineering helps the construction of more reliable software, mostly through formal verification. One widely used approach to formal verification is model checking – an automated technique that verifies whether a property specified in a temporal logic is satisfied in a finite state model of the system, given a particular initial state [Baier and Katoen 2008] [Grumberg et al. 1999]. Under such a perspective, we can verify self-adaptive properties in SAS finite state models to automatically generate adaptive solutions through a model checking planning problem characterization in consonance with MAPE-K.

In this work, we characterize the generation of autonomous plans as a planning problem using a study case to investigate autonomous adaption plans with model checking. We present some results towards an adaptation planning solution in SAS which autonomously direct their self adaptation from defined scenarios at run-time.

The purpose of this work is to show how to design SAS that satisfy several self-*properties by the generation of adaptation plans autonomously using model checking. Although MAPE-K is widely used in SAS projects, it has been little explored in formal models. Our work proposes a solution for SAS specification based on the MAPE-K architecture, in which it is possible to add/remove adaptation objectives by easily plugging/unplugging self-*properties controllers from the formal model.

The remaining of this paper is organized as follows. Section 2 presents related work, while Section 3 presents self-adaptive concepts and the MAPE-K reference model. Section 4 introduces planning with model checking, Section 5 discusses the case-study about a decentralized traffic monitoring system model checking-based planning and Section 6 shows the results of the work. Finally, Section 7 draws the conclusions and opportunities of future research.

2. Related Work

In this section, we present some related work on SAS model checking approaches, MAPE-K-based SAS, arbitrary planning from goals specified in a temporal logic, and planning based on model checking.

In [Iftikhar and Weyns 2012, Weyns 2012], a case-study consisting of a decentralized traffic monitoring SAS is presented. A formal architecture model is proposed and a group of self-adapting properties are verified using model checking. youThe following properties are checked: flexibility expressing the ability of the system to dynamically adapt in response to changing conditions in the environment; and robustness expressing system's ability to autonomously deal with errors during execution are verified. Such properties guarantee the self-healing behavior when a failure occurs in the system. However, the study does not focus on all failure scenarios and on planning strategies for adaptation. Also, it does not use MAPE-K as a reference model. Our work extends this study addressing such issues.

In [Arcaini et al. 2017], the authors present a conceptual and methodological framework for formal modeling, validation, and verification of distributed SAS. The framework supports formal techniques to validate and verify adaptation scenarios and to obtain feedback (at design-time) concerning the correctness of the implementation of the adaptation logic with MAPE-K loops. [Camilli et al. 2015] presents a formal approach to specify and verify SAS behaviors in real-time using time-based Petri Nets. In [Iglesia and Weyns 2015], the authors present a set of formal MAPE-K models that comprise behavior specification models of different components of a MAPE-K feedback loop and their interactions, as well as adaptive property specification templates for behaviors formal verification. They define templates for behavior specifications and properties, concluding that their templates have a balance between generality and usability. In both studies the planning problem for adaptation is not addressed.

In [Sykes et al. 2007, Sykes et al. 2008] the authors consider the challenges for providing a standalone system with the ability to direct its own adaptation. They describe an initial implementation where changes in the software architecture is executed as a result of the execution of a reactive plan. The adaptation is done at run-time in software component configurations that compose the system, according the operations they execute. The system allows arbitrary dynamic reconfiguration, exploring the presence of a reactive plans determining the system behavior. Reactive plans are generated with a planning tool from goals provided by the user, given in CTL [Behrmann et al. 2006] [Grumberg et al. 1999] temporal logic. A set of condition rules indicates which components will be required to execute the plan. Likewise, our study also considers that an autonomous system must have the ability to direct its own adaptation but we focus in SAS in general and not on component-based software architecture. We consider the process of autonomous generation of an adaptation plan with model checking but we do not use those plans for dynamic system reconfiguration.

In [Giunchiglia and Traverso 1999], the authors present planning problems can be stated as model checking problems, that formalization can be used to tackle various planning problems, and how planning as model checking can be implemented. This work provides a general planning approach and have influenced our investigation about planning in SAS.

3. Self-Adaptive Systems

SAS evaluate their own behavior, modifying themselves when their primary purpose is not being effectively fulfilled or a better functionality and/or performance can be achieved

[Laddaga 1997]. Such systems usually adopt a model of themselves and their environment and adapt their structure and/or behavior as necessary, according to some adaptation goals [Oreizy et al. 1998].

Andrade [Andrade 2014] listed five motivations for a system to adapt itself: *i)* increasing complexity of business rules (problem space), which demands automatic implementation/configuration of solutions; *ii)* data dynamism, characterized by constant changes in the nature of the data; *iii)* dynamism in service demands; *iv)* dynamism in the execution environment caused by intentionally partial or uncertain requirements; and *v)* inherent self-adaptiveness of scenarios, problem or software's own functional structure. In these cases adaptive solutions are desired in order to ensure the system working properly in response to environment changes [Salehie and Tahvildari 2009a].

Decentralization is often a feature of cooperative self-adaptive or self-organizing systems, which work with no central authority [Wildermann 2012]. They usually have a large number of components which interact locally driven by simple rules. The global behavior of the system emerges from such local interactions. A system with many interacting elements that are either absolutely unaware or have only partial knowledge about the global system is named a Self-Organizing System [Salehie and Tahvildari 2009b]. Such systems are characterized by a decentralized control mechanism [Wildermann 2012] and autonomous/decentralized cooperation of agents [Andrade 2014].

A Self-healing System is a kind of SAS characterized by being aware of its state and having the ability to detect and recover itself from known or unknown faults [Momeni et al. 2008]. SHS has the ability to discover, diagnose, and react to system malfunctioning [Salehie and Tahvildari 2009b] [Al-Zawi et al. 2011], anticipating possible problems and, therefore, take appropriate actions to avoid failures [Salehie and Tahvildari 2009b].

SAS usually implements some kind of feedback loop, which continuously evaluate the system and the environment, decides about the need of adaptations, and enacts those changes in the system when appropriate [Naqvi 2012]. A SAS comprises two parts: the managed system – which deals with the functionality of the application domain; and the manager system – which addresses managed system adaptations to achieve specific quality objectives [Weyns et al. 2012]. The manager system is one that implements one or more feedback loops.

The feedback loop is needed to monitor the system itself and its context, detecting significant changes, deciding how to react, and acting to execute adaptations [Salehie and Tahvildari 2009b]. A feedback loop MAPE-K is design to perform the following tasks [Iglesia and Weyns 2015][Kephart and Chess 2003]:

- `Monitor`: collects system and environment data to update knowledge information. Monitoring involves the capture of environmental data important for the self-properties of the system;
- `Analyze`: determines whether adaptation actions are required based on monitoring data;
- `Plan`: puts together an action plan which drives the system back to its expected level of service. It takes into account the monitoring data to produce a set of changes to be made in the managed system;

- `Execute`: enacts adaptation actions provided by the generated plan;
- `Knowledge`: provides an abstraction of the managed system and the environment in which the SAS operates, describing the knowledge related to the questions of adaptation and representing information of run-time aspects.

4. Planning with Model Checking

Planning is an abstract and explicit deliberation process that chooses and organizes actions in anticipation from expected outcomes. Planning involves devising a plan of actions to achieve one's goals. Automated Planning is a field of Artificial Intelligence (AI) that studies this process computationally [Ghallab et al. 2004] in order to elaborate plans for a computational agent to reach its goals [da Silva Ferreira et al. 2018].

A planning approach based on formal methods is very attractive to ensure reliable solutions [Pereira and de Barros 2008]. Giunchiglia and Traverso (1999) defines a planning problem as a model checking problem, where a planning domain is modeled as a state space model and a planning goal is specified as a property in a temporal logic. According to this approach, an adaption plan is achieved by verifying whether a temporal formula which specify an adaptation goal is true in the planning state space when evaluated from a given initial state. The solution for a planning problem is a set of plans, each one consisting of a sequence of actions that starts from an initial state to a satisfaction state which represents the planning goal [da Silva Ferreira et al. 2018].

In this work, we adopt the Computation Tree Logic (CTL) formalism to specify adaptation goals. We reason with CTL on the planning state space, which is a finite transition graph that can be unfolded from an initial state in an tree with a finite set of paths, quantifying over such paths. A plan can be defined from a path which satisfies the adaption goal. The CTL syntax is given by the Backus-Naur form [Huth and Ryan 2004]:

$$\phi ::= \top|\bot|p|(\neg\phi)|(\phi \vee \phi)|(\phi \wedge \phi)|(\phi \rightarrow \phi)|$$

$$EX\phi|AX\phi|EF\phi|AF\phi|EG\phi|AG\phi|E[\phi \cup \phi]|A[\phi \cup \phi]$$

where p is an atomic propositional formula and $\neg$, $\wedge$, $\vee$, and $\rightarrow$ are propositional connectives. The temporal operators X, F, G and U, mean "next state", "eventually in the future state", "in all states (globally)" and "until a state", respectively. Each one must be preceded by a path quantifier A or E, where A means "for all paths" (inevitably) and E means "exist a path".

Planning with model checking allows the automatic generation of plans by controlling the evolution of the system so that system behaviors can satisfy the goal [da Silva Ferreira et al. 2018].

Definition 1: Planning Domain. A planning domain is a Labelled Transition System (LTS) $\mathcal{D}$ = (S, I, A, $\longrightarrow$, AP, L), where:

- S is a finite set of states;
- I $\subseteq$ S is the set of initial states;
- A is the actions alphabet;
- $\longrightarrow \subseteq S \times A \times S$ is the set of transitions state relations;
- AP is a fixed and finite set of atomic propositions used to label the states;

- L: S $\longrightarrow 2^{AP}$ is a total function that assigns to each state s $\in$ S the set L(s) of atomic propositions which are true in that state.

Definition 2: Planning Problem. A planning problem P in a planning domain $\mathcal{D}$ = (S, I, A, $\longrightarrow$, AP, L) is a tuple $\langle \mathcal{D}, s_p, \phi \rangle$, where [Ghallab et al. 2004]:

- $\mathcal{D}$ is the planning domain;
- s_p is the planning problem's initial state, where $s_p \in S$;
- ϕ is the adaptation goal, specified in temporal logic.

Definition 3: Plan. A plan in a planning domain $\mathcal{D}$ = (S, I, A, $\longrightarrow$, AP, L) for a planning problem $\mathcal{P} = \langle \mathcal{D}, s_p, \phi \rangle$ is defined as a sequence $\pi = [\langle s_0, a_1 \rangle, \langle s_2, a_2 \rangle ... \langle s_k, a_k \rangle]$, where $s_0 = s_p, s_i \in S, a_i \in A, 0 \leq i \leq k$ and s_k satisfies ϕ, i.e. s_k is the goal state.

Definition 4: Planning Solution. A planning solution in a planning domain $\mathcal{D}$ = (S, I, A, $\longrightarrow$, AP, L) for a planning problem $\mathcal{P} = \langle \mathcal{D}, s_p, \phi \rangle$ is a subgraph $\mathcal{D}_\pi$ of $\mathcal{D}$ such that in $\mathcal{D}_\pi$ all paths from the state s_p contain a plan which satisfies the goal ϕ.

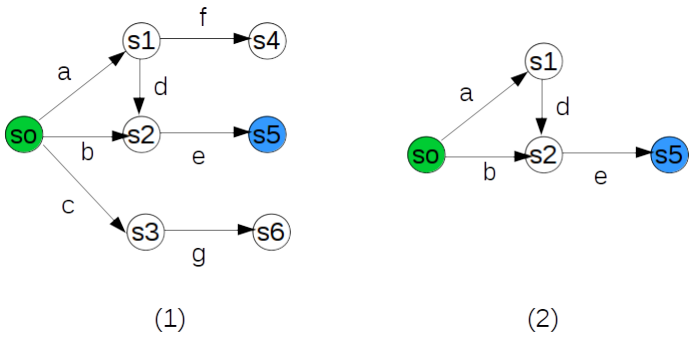

Figura 1. Planning solution (adapted from [Pereira and de Barros 2008]).

Definition 5: Deterministic Planning Problem. A planning problem $\mathcal{P} = \langle \mathcal{D}, s_p, \phi \rangle$ is deterministic if only if there is only one plan π in the planning solution $\mathcal{D}_\pi$.

For example, in Fig. 1(1), the green state is state inicial and blue state is goals state, we have a planning domain $\mathcal{D}$ modeled by a LTS with the set of states $S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$, the set of initial states $I = \{s_0\}$ and the set of actions $A = \{a, b, c, d, e, f, g\}$. The planning problem solution from the initial state s_0 to a goal state s_5 is represented in Fig. 1(2) as the domain $\mathcal{D}_\pi$ which has the plans $\pi_1 = \{\langle s_0, b \rangle, \langle s_2, e \rangle\}$ and $\pi_2 = \{\langle s_0, a \rangle, \langle s_1, d \rangle \langle s_2, e \rangle\}$.

Several factors influence the choice of an adaptation plan. Some of these factors are related to the environment that can be deterministic or non-deterministic. In a deterministic environment next states are completely determined by the current state and the actions performed by the agent; otherwise, it is non-deterministic [Russell and Norvig 2016].

Given a planning problem, we can distinguish three classes of solutions for non-deterministic environments: weak, strong and strong-cyclic; each one indicating a different quality of planning [Pereira and de Barros 2008].

Considering a planning problem $\mathcal{P} = \langle \mathcal{D}, s_0, \phi \rangle$, in a domain $\mathcal{D}$ = (S, I, A, $\longrightarrow$, AP, L), a planning solution can be [Pereira and de Barros 2008]:

- A weak solution for $\mathcal{P}$, if not all path in $\mathcal{D}$ induce a plan in $\mathcal{D}_\pi$;
- A strong solution for $\mathcal{P}$, if all plans in $\mathcal{D}_\pi$ are acyclic;
- A strong-cyclic solution for $\mathcal{P}$, if in all path in $\mathcal{D}$ there is a plan in $\mathcal{D}_\pi$.

A weak planning solution can allow the system to achieve the goal state but it is not guaranteed due to the non-determinism of the environment. A strong solution always achieves the goal state ensuring that a plan is executed, regardless of non-determinism of the environment. In a strong-cyclic solution, the goal is achieved under the fairness assumption that execution will eventually exit regardless the existence of cycles in the domain model [Pereira and de Barros 2008].

Given $\mathcal{G}$ a propositional CTL formula representing an adaptation goal, we can look for a weak plan by the formula EF $\mathcal{G}$. A strong plan should satisfy the formula AF $\mathcal{G}$ and a strong cyclic plan should satisfy the formula AGEF $\mathcal{G}$ [Giunchiglia and Traverso 1999].

5. Case-Study

Our work is build on top of the case-study of Iftikhar and Weyns [Iftikhar and Weyns 2012] which consists of a decentralized traffic monitoring SAS with groups of three cameras. Model checking is used to verify some behavioral properties and self-adaptive properties for flexibility and robustness, but not focusing on planning. We extend this case study considering groups of two to five cameras where – for each group – a camera is added to a group when identifying traffic jam or is recovered from a fail and a camera is removed from a group when it fails or when it does not identify more traffic jam. With that, self-adaptation scenarios grown in complexity allowing us to state such an arrangement as a planning problem, i.e., adaptation plans are generated to reconfigure groups of cameras when cameras fail/recover or they detect traffic jam / not traffic jam.

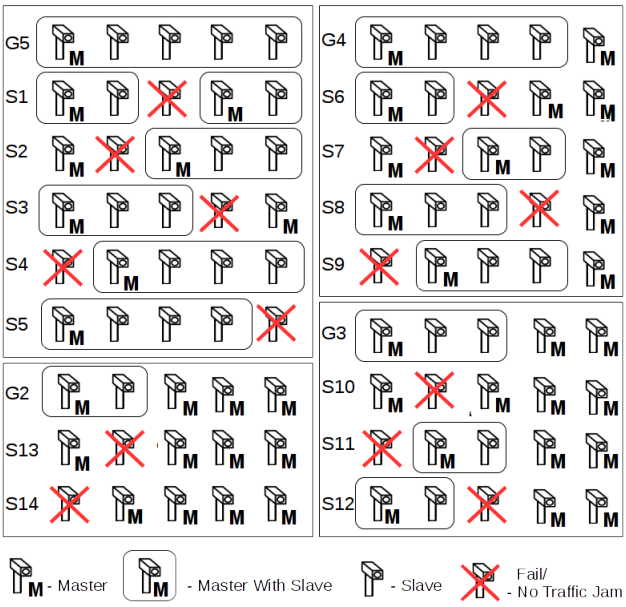

Figura 2. Cameras Groups Scenarios.

The cameras are distributed along a road and each one has a limited viewing range and is positioned to provide optimal highway coverage with minimal overlap. To have

a decentralized solution, the cameras collaborate in group organizations: if a traffic jam goes beyond the reach of a camera this camera will form a group with another neighboring camera, which in turn also identifies the traffic jam. When a group of cameras is formed each camera monitors if its neighboring camera is properly working.

5.1. Scenario

This study addresses the removal and addition occurrence of cameras in groups formed by 2, 3, 4, and 5 cameras, according to Fig. 2. G2, G3, G4 and G5 represent such groups before failures occurrence, respectively. Each working camera is in one of three distinct roles: (1) a master of a single-member organization; (2) a master of an organization with slaves; or (3) a slave in an organization. Each role has the corresponding responsibilities: (1) a master of a single agent organization does not monitor any camera and is not monitored; (2) a master with slaves in a group monitors the last camera in the group; (3) a slave of a group monitors its left neighboring camera.

For each possible group, the behavior of the system was analyzed when each one of the cameras failed or did not identify any more traffic jam (we assume only one camera failed at a time). We also analyzed the behavior of the groups when each one of the cameras recovered from fail or began to identify traffic jam again. All scenarios with more than 5 cameras will fit into some scenario predicted in this study. With groups of two to five cameras it was possible to map 14 possible scenarios of self-organization caused by cameras removal and others 14 possible scenarios of self-organization caused by addition of cameras in a group.

5.2. Modeling

In this work, we have used UPPAAL tool to support planning with model checking. UPPAAL is a model checker based on timed automata theory which uses TCTL (Timed Computation Tree Logic) to specify properties to be checked [Behrmann et al. 2006].

A model in UPPAAL is represented as a directed graph, with nodes and transitions, such that from each node there is an outgoing transition. In this paper, we refer to the nodes as states and we refer to a state x as the node named with x and sometimes also the committed nodes (one with a c inside it) between x and another named node. For example, in model of Fig. 3 when we refer to SLAVE state we mean the node named SLAVE and the other two committed nodes which are in the cycle to SLAVE state. Committed nodes are used to model atomic actions and we use them to model adaptation plans such that when an adaptation plan is in execution it is not be interrupted by camera failures or other actions. The initial state of the model is the node with two concentric circles. The expressions in the transitions can be a boolean expression (guards) which expresses a condition to a transition be triggered, internal actions (assignments or function calls), or synchronizing actions ($e[m]!$ and $e[m]?$) where $e[m]!$ denotes that the message m is sent by the the communication channel e and $e[m]?$ denotes that the message m is received from e setting a handshaking communication between models.

MAPE-K based modeling is divided into manager system and managed system. The managed system in our case-study consists of the organizing controller (Fig. 3), thus forming the system domain. The manager system consists of MAPE-K self-healing controller (Fig. 4) and MAPE-K self-organizing controller (Fig. 5). The model that

simulates the operation of the system by triggering traffic jam and faults and other models that support the solution are omitted from this article. We extended the from [Iftikhar and Weyns 2012] case study in Fig. 3 and Fig. 4.

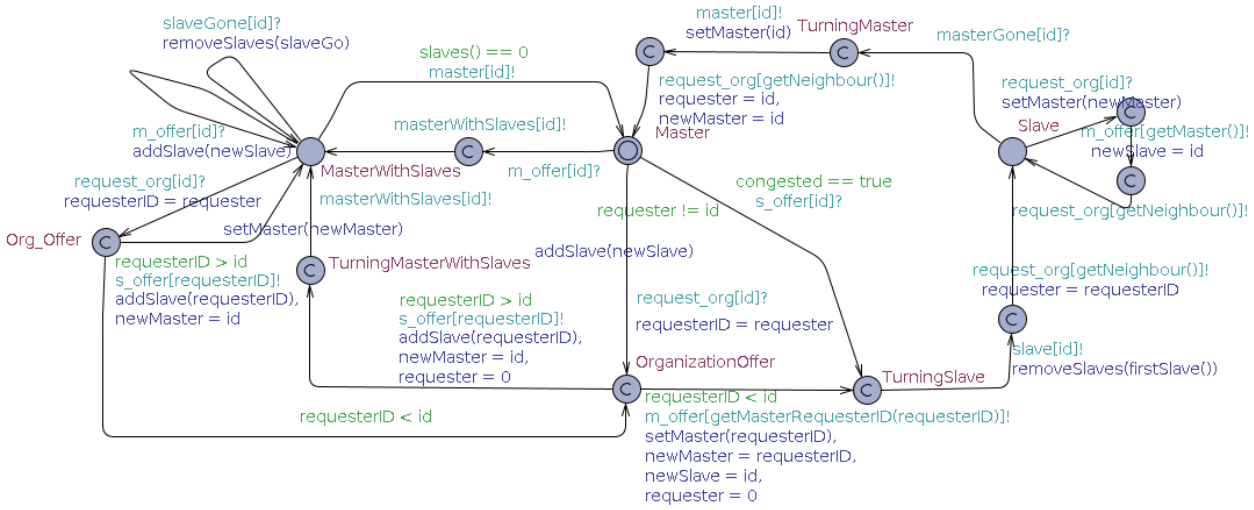

Figura 3. Organizing Controller.

Managed System: The organizing controller model (Fig. 3) is responsible for defining the cameras behaviours to be (re) organized into groups. The (re) grouping starts on a camera, and this camera can request its neighboring camera to be re (organized) as well. For example, a camera in the MASTERWITHSLAVE state can move to MASTER state when no longer it has slaves, as for example in case of scenario 13 of Fig. 2; or it can move to SLAVE state if a new master camera with smaller id requests a new organization, as in case of scenario 4 of Fig. 2 when the camera 1 is recovered.

A camera can play three different roles. The choice of which role to assume does not depend exclusively on the camera itself. The dynamic environment allows it to be a MASTER and depending on its neighboring camera it becomes a MASTERWITHSLAVE or a SLAVE (Fig. 3), and this can only be determined at run time. In addition to these roles, a camera may be in the state FAILED (Fig. 4) indicating that a camera has not returned a ping and then a plan should be executed so that the system continues to function properly.

manager system: The manager system is represented by MAPE-K Self-healing Controller and MAPE-K Self-organizing Controller models. For each controller a feedback loop was implemented. The monitor of the self-healing controller (Fig. 4) has an observer loop that detects failures, and implements the ping-echo mechanism by sending life signals to other observed cameras. The MAPE-K analysis task is done from the (WAITFORRESPONSE state in Fig. 4) by checking if the response time of the neighboring camera is higher than expected ($X > ALIVE\_TIME$). If so, it changes to FAILUREDETECTED state, indicating that a camera in the group failed and an adaptation is required. The Knowledge (implemented as variables in UPPAAL) has information about the formation of the group, the camera that failed (ping) and the camera that identified the failure (id). For the self-organizing controller (Fig. 5) the MAPE-K monitor task

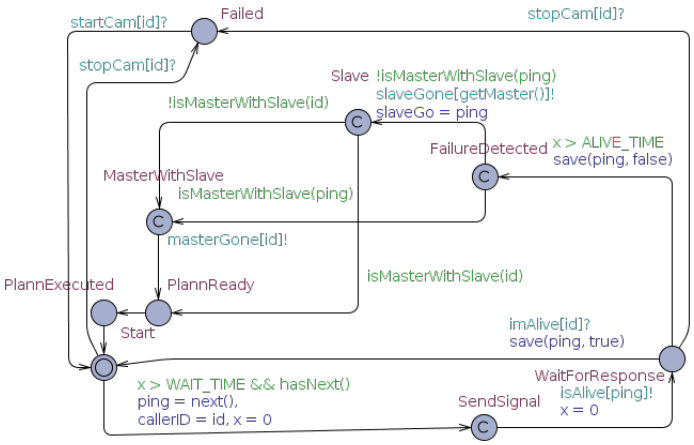

Figura 4. MAPE-K Self-Healing Controller.

was implemented as a loop that collects traffic jam data. The MAPE-K analysis verifies (ANALYZINGCONGESTION state) whether the camera identifies a change in traffic jam information. If so, it goes to CHANGECONGESTION state, indicating that an adaptation is required. The Knowledge has information about whether the state is CONGESTION-DETECTED or NOCONGESTION. In the first case, it is required a (re) organization of the group. In the second case, the camera ceases to be a master (MASTERWITHSLAVE state) or ceases to be a slave (SLAVE state) forcing a search for a plan in order to create a new group. For both models an adaptation plan is dynamically found through verification of properties verification over the planning domain model. In the PLANEXECUTED state an adaptation plan is executed.

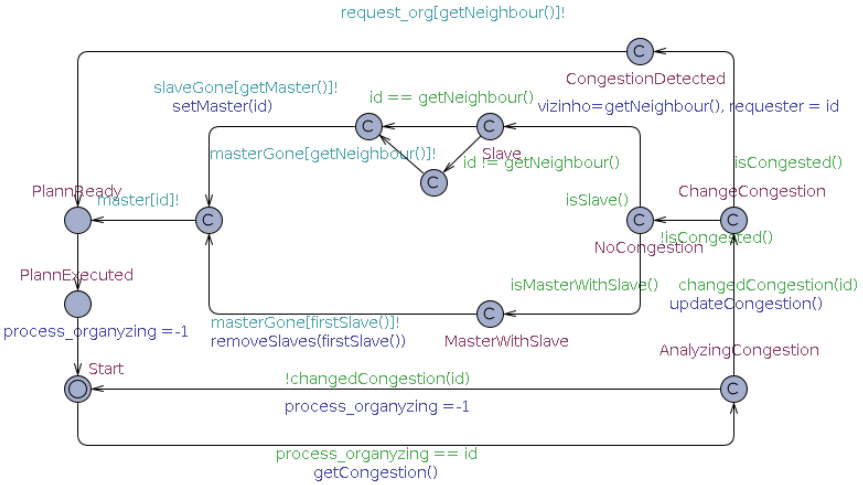

Figura 5. MAPE-K Self-Organizing Controller.

5.3. Verification

An adaptation goal, for this case study, must ensure that the adaption plan execution reconfigures the camera groups so that each group has a camera MASTERWITHSLAVE and at least one camera SLAVE. For this, two properties were specified in TCTL as can be seen in Fig. 6.

Property (1) and (2) check, for the case of camera failure and traffic jam detection, respectively, whether after the execution of an adaptation plan the reconfiguration of the groups is as expected according scenario S1, considering that the cameras are initially in configuration G5. It means that for all paths and all states from the current state PLANEXECUTED (Fig. 4) the camera 1 has cameras 2 ($camera[2].m\_cam == 1$ in Fig. 6) as its slaves and camera 4 has camera 5 ($camera[5].m\_cam == 4$ in Fig. 6) as its slave. This property specific for scenario S1 might be adapted to other scenarios. Indeed we verified that in all scenarios the reconfiguration of the groups was as expected.

Property (3) specifies that every camera which identifies a failure, after executing a plan, becomes a MASTER or a MASTERWITHSLAVE. Finally, the property (4) guarantees that after the execution of the plans the camera that became MASTERWITHSLAVES necessarily owns slaves. We verified that property (3) and (4) were valid in all scenarios from PLANEXECUTED state in the model of Fig. 4.

```
A[] (SelfHealingController(4).PlannExecuted  && Camera(3).Failed)
imply (Camera(1).MasterWithSlaves && camera[2].m_cam == 1) &&
(Camera(4).MasterWithSlaves && camera[5].m_cam == 4)                    (1)

A[] ((SelfOrganizingController(3).PlannExecuted and
Camera(3).Master) imply (Camera(1).MasterWithSlaves &&
camera[2].m_cam == 1) && (Camera(4).MasterWithSlaves &&
camera[5].m_cam == 4))                                                  (2)

A[] forall (n: cam_id) (SelfHealingController(n).PlannExecuted  imply
Camera(n).Master or Camera(n).MasterWithSlaves)                         (3)

A[] forall (n: cam_id) ((SelfOrganizingController(n).PlannExecuted and
Camera(n).MasterWithSlaves) imply camera[n].totalSlaves >0)             (4)
```

Figura 6. Properties.

6. Case Study Considerations

The generation of autonomous adaptation plans can be seen as a planning problem. According to the planning problem definition presented in Section III, the planning domain $\mathcal{D}$ comprises the managed system and manager system models. The initial state s_0 is the FAILUREDETECTED state of the self-healing controller model (Fig. 4) of camera 4 which detected the fault. The adaptation goals ϕ are the temporal logic formulas (1) and (3) (Fig. 6).

We can observe that for each verified scenario there is only one adaptation plan that meets the specified adaptation goals ϕ specified in CTL which implies that, in accordance with definition 5, the planning problem of this case study is deterministic.

Adaptation plans will only be defined at runtime because the current state of the camera that identifies a change is not sufficient to carry out a plan. It is necessary to know the global behavior (cameras group) which is identified by interactions between local behaviors (individual cameras).

So, for the generation of a plan it is necessary for a camera to know the current state of its neighboring camera which in turn must known about its neighboring current state, and so on. For example, if a camera which is a slave failed and the camera that has

identified it is also a slave different plans can be executed. In scenario S1 of Fig. 2, for example, two groups were formed, in scenario S2 only one group with only the master with slaves camera that identified the fault with two slaves and in scenario S3 a group with the master camera which identified the fault with two slaves.

For illustrating the plan generation, we consider scenario S1 with one master with slave and four slaves where camera 3 which is a slave failed. The verification of properties (1) and (3) generated the following plan, with 13 actions:

$1.\langle FailureDetected, slaveGone[getMaster()]!\rangle$ $2.\langle MasterWithSlave, slaveGone[id]?\rangle$
$3.\langle SLAVE, masterGone[id]!\rangle$ $4.\langle SLAVE, masterGone[id]?\rangle$
$5.\langle TurningMaster, master[id]!\rangle$ $6.\langle TurningMaster, request\_org[getNeighbour()]!\rangle$
$7.\langle Slave, request\_org[getNeighbour()]?\rangle$ $8.\langle Slave, m\_offer[getMaster()]!\rangle$
$9.\langle Master, m\_offer[id]?\rangle$ $10.\langle Master, masterWithSlave[id]!\rangle$
$11.\langle Master, masterWithSlave[id]?\rangle$ $12.\langle Slave, request\_org(getNeighbour())!\rangle$
$13.\langle PlanReady, null)\rangle$

The plans generation occur atomically, i.e, we did not consider the occurrence of camera failures when a plan is in execution. This can at first be seen as a limitation of our case study, but this restriction was considered for circumvent state space explosion problem, inherent of model checking, enabling us to achieve our main objective which was to demonstrate the planning strategy in operation.

In this case study we specify a formal model fully compatible with MAPE-K reference architecture where the behavior of the manager system is separated from the behavior of the managed system. Thus, the adaptations are controlled by MAPE-K feedback loops, coupled with system domain, making it more flexible to add / remove new adaptations due to less cohesion between functionalities.

Using MAPE-K architecture, we encapsulated the plans selection strategies in controller components, leaving the execution of the plans to the system domain. That is, the adaptation plans belonged to the system domain itself and a MAPE-K controller selected the adaptation plan autonomously. With this, we defined a flexible formal model MAPE-K where planning strategies can be inserted or removed.

7. Conclusion

In this paper we presented some results of a SAS planning case study, consisting of a decentralized traffic monitoring, in which we combined model checking and MAPE-K in order to find adaptation plans autonomously when interruptions caused by failures occur or when changing in traffic jam requires re (organization) of groups of cameras.

Model checking has shown to be promising to generate adaptation plans autonomously, when the number of possible adaptation plans is great. Formal modeling of SAS based on the MAPE-K architecture contributes for a reliable design of SAS, facilitating the generation of adaptation plans that meet more than one self- * property (for example, Self-Healing and Self-Organizing), since a feedback loop can be associated for each self-*property.

The paper presents an experience of using model checking for the generation of adaptation plans, which can be taken as an example to guide SAS design in order to generate adaptation plans autonomously satisfying more than one self- * property.

Our approach needs to be further studied in other domains and in other type of environments since our case study considered a deterministic planning problem. So, we intend to extend it to non deterministic planning problems.

As far as we know, few studies focus in development of self-adaptive systems planning with model checking and MAPE-K. We believe that the dynamic environment of these systems discourages the use of model checking for planning due to state explosion problem and modeling complexity. However, MAPE-K enabled modularization such that we could focus on modeling aspects related to the adaptation plan rather than the entire system, bypassing the state explosion, and also facilitating modeling. These results encourage we are investigating a general method for planning of self-adaptive systems with model checking at run time, which we have already begun to investigate.

Referências

Al-Zawi, M. M., Al-Jumeily, D., Hussain, A., and Taleb-Bendiab, A. (2011). Autonomic computing: Applications of self-healing systems. In *2011 Developments in E-systems Engineering*, pages 381–386. IEEE.

Andrade, S. S. (2014). Projeto arquitetural automatizado de sistemas self-adaptive–uma abordagem baseada em busca.

Arcaini, P., Riccobene, E., and Scandurra, P. (2017). Formal design and verification of self-adaptive systems with decentralized control. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 11(4):25.

Baier, C. and Katoen, J.-P. (2008). *Principles of model checking*. MIT press.

Behrmann, G., David, A., and Larsen, K. G. (2006). A tutorial on uppaal 4.0. *Department of computer science, Aalborg university*.

Berns, A. and Ghosh, S. (2009). Dissecting self-* properties. In *2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 10–19. IEEE.

Camilli, M., Gargantini, A., and Scandurra, P. (2015). Specifying and verifying real-time self-adaptive systems. In *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, pages 303–313. IEEE.

da Silva Ferreira, M., Menezes, M. V., and de Barros, L. N. (2018). Plan existence verification as symbolic model checking. In *Anais do XV Encontro Nacional de Inteligência Artificial e Computacional*, pages 116–127. SBC.

Ghallab, M., Nau, D., and Traverso, P. (2004). *Automated Planning: theory and practice*. Elsevier.

Giunchiglia, F. and Traverso, P. (1999). Planning as model checking. In *European Conference on Planning*, pages 1–20. Springer.

Grumberg, O., Clarke, E., and Peled, D. (1999). Model checking.

Huth, M. and Ryan, M. (2004). *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge university press.

Iftikhar, M. U. and Weyns, D. (2012). A case study on formal verification of self-adaptive behaviors in a decentralized system. *arXiv preprint arXiv:1208.4635*.

Iglesia, D. G. D. L. and Weyns, D. (2015). Mape-k formal templates to rigorously design behaviors for self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 10(3):15.

Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *Computer*, (1):41–50.

Laddaga, R. (1997). Darpa broad agency announcement on self-adaptive software.

Momeni, H., Kashefi, O., and Sharifi, H. (2008). How to realize self-healing operating systems? In *2008 3rd International Conference on Information and Communication Technologies: From Theory to Applications*, pages 1–4. IEEE.

Naqvi, M. (2012). Claims and supporting evidence for self-adaptive systems–a literature review.

Oreizy, P., Medvidovic, N., and Taylor, R. N. (1998). Architecture-based runtime software evolution. In *Proceedings of the 20th international conference on Software engineering*, pages 177–186. IEEE.

Pereira, S. L. and de Barros, L. N. (2008). A logic-based agent that plans for extended reachability goals. *Autonomous Agents and Multi-Agent Systems*, 16(3):327–344.

Russell, S. J. and Norvig, P. (2016). *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.

Salehie, M. and Tahvildari, L. (2009a). Self-adaptive software: Landscape and research challenges. *ACM transactions on autonomous and adaptive systems (TAAS)*, 4(2):14.

Salehie, M. and Tahvildari, L. (2009b). Self-adaptive software: Landscape and research challenges. *ACM transactions on autonomous and adaptive systems (TAAS)*, 4(2):14.

Sykes, D., Heaven, W., Magee, J., and Kramer, J. (2007). Plan-directed architectural change for autonomous systems. In *Proceedings of the 2007 conference on Specification and verification of component-based systems: 6th Joint Meeting of the European Conference on Software Engineering and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 15–21. ACM.

Sykes, D., Heaven, W., Magee, J., and Kramer, J. (2008). From goals to components: a combined approach to self-management. In *Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*, pages 1–8. ACM.

Weyns, D. (2012). Towards an integrated approach for validating qualities of self-adaptive systems. In *Proceedings of the Ninth International Workshop on Dynamic Analysis*, pages 24–29. ACM.

Weyns, D., Iftikhar, M. U., De La Iglesia, D. G., and Ahmad, T. (2012). A survey of formal methods in self-adaptive systems. In *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*, pages 67–79. ACM.

Wildermann, S. (2012). Systematic design of self-adaptive embedded systems with applications in image processing.