Design and Timed Verification of Self-adaptive Systems

Marwa Hachicha, Riadh Ben Halima and Ahmed Hadj Kacem
University of Sfax, ReDCAD Laoratory, B.P.1173, Sfax-Tunisia
Email: marwa.hachicha@redcad.org, riadh.benhalima@enis.rnu.tn, ahmedhadjkacem@fsegs.rnu.tn

***Abstract*—Self-adaptive systems are able to manage themselves autonomously. A common approach to engineer these systems is to use the MAPE control loop based on these four steps: Monitoring, Analysis, Planning, and Execution. Existing research pays little attention to the modeling of self-adaptive systems with multiple MAPE control loops, and is lacking in considering the formal specification and verification of temporal constraints in such systems. In this paper, we present a new approach for modeling self-adaptive systems with multiple MAPE control loops and we present a set of time patterns for self-adaptive systems. We illustrate our approach by modeling and verifying a time critical forest fire detection system that exhibits a self-adaptive behavior.**

I. Introduction

Self-adaptive systems regulate system characteristics in response to a system or environmental state change. J.O.Kephart et al. [1] proposed the notion of an autonomic element, in the form of a Monitoring-Analysis-Planning and Execution (MAPE) loop that controls a managed computing element.
In order to ensure the correctness of the system's adaptation logic, the execution of the different self-adaptive system events can be constrained by several properties such as temporal properties. The time dimension in self-adaptive systems is of paramount importance since the different non-functional properties can be time-depend. For example, in safety critical real-time systems such as crisis management system, rail-road crossing systems, or flight control, a slow response that does not meet the time constraints could result in big damages.
However, the survey in [2] shows that formally founded design models that verify relevant properties for self-adaptive systems are highly demanded.
In the other hand, when systems are large, complex, and heterogeneous, a single MAPE loop may not be sufficient for managing all adaptation in a system, so multiple MAPE loops may be introduced. Consequently, modeling, specifying and verifying complex self-adaptive systems will be too complicated especially when involving time constraints. In this paper, we propose a refinement based approach to model and verify time constraints for self-adaptive systems. Our approach is divided into three phases:

Firstly, we propose modeling self-adaptive systems that instantiate the well-known MAPE architectural patterns proposed by [3]: master/slave, coordinated control, hierarchical control, regional planning and information sharing. These patterns model different types of interacting MAPE loops in self-adaptive systems with different degrees of decentralization.

Indeed, after designing step-by-step structural and behavioral features of self-adaptive systems, the expert has to ensure the system performance by the verification of several properties. In this paper, we concentrate on the verification of time constraints for self-adaptive systems.

So, in the second step, we propose an automatic transformation of the designed self-adaptive system into Event-B [4] specifications. The translation to Event-B is achieved through many refinement steps while using a set of transformation rules. In other words, refinement allows us to build a model gradually by making it more and more precise.

Thirdly, the resulting model is enriched by different temporal properties. In this context, we propose a support of time pattern for self-adaptive systems based on the Event-B method. A property that has been proved as correct in the first level must still be correct in the subsequent level.
The remainder of this paper is organized as follows. In section 2, we provide some background information on Event-B method,and we introduce the forest fire detection system as our case study to illustrate our proposed approach. In section 3, we expose the modeling phase of our self-adaptive system. In section 5, a description of the different time patterns to check temporal violation of self-adaptive system process is presented. Section 6 presents a survey of related work. Finally, the last section concludes the paper and gives future work directions.

II. Background Concept

This section briefly introduces the Event-B method, MAPE patterns for self-adaptive systems and our case study example.

A. Event-B method

The Event-B [4] modeling language defines mathematical structures into contexts and formal model of the system into machines. The context is defined by abstract sets, constants, and axioms which describe properties of constants. An Event-B machine describes a reactive system by a set of invariant properties and a finite list of events modifying state variables.

A machine M may see a context C, this means that all carrier sets and constants defined in C can be used in M. A context Ć can extends a context C, this means that all properties defined in C are added to Ć.

Event-B uses a top-down refinement-based approach. The refinement of a specification allows enriching it in a step-by-step fashion.

978-1-5090-5507-4/17/$31.00 ©2017 IEEE
ICIS 2017, May 24-26, 2017, Wuhan, China

B. Forest Fire Detection System (FFDS) use case

In the last couple of years, large wildfires have caused extended damages and catastrophic consequences in lost of properties and lives. Consequently, early detection and suppression of fires deem crucial. We propose a self-adaptive forest fire detection system, designing by the Ruer Bokovi Institute, and comprising a number of Peripheral Observation Point (POP) units installed at sites and supporting meteorological sensors. These POP are linked via radio or wire cables to a Command and Control Center (CCC). The CCC merges all information from observation point and analyzes them to achieve reliable and efficient fire detection. These modules can be distributed over the territory so as to cover the entire area of interest and can be connected to a hierarchically higher control which acts as Regional supervisor. When fire was detected, the regional planner takes possible adaptation actions using effector such as dispatching fire fighting crews and sending unnamed aerial vehicles (UAV).

To simplify modeling, our forest fire detection system is based on three POP supporting three sensors (humidity, gas and temperature), two CCC and two regional supervisors.

If the FFDS fails to detect fire within a specific time, this is can cause significant damage to natural and human resources. Therefore, checking that all the temporal constraints in processing the different events are met is very crucial.

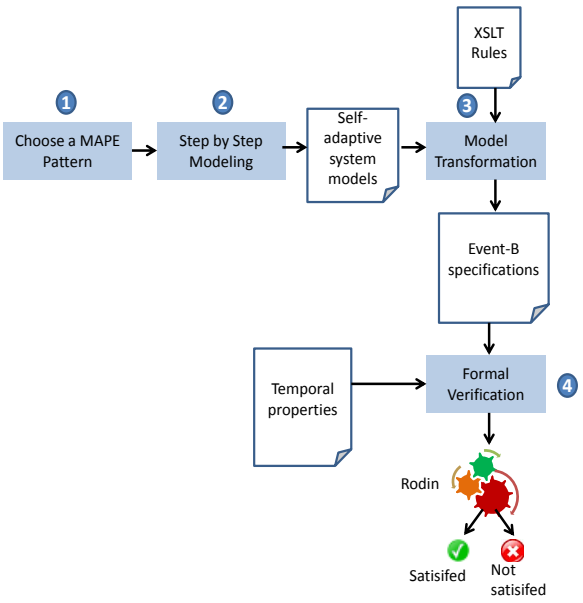

Fig. 1. Proposed Approach

III. Step-By-Step Self-Adaptive Systems Modeling

As presented in Fig.1, the first step of our approach is the step-by-step modeling for describing self-adaptive systems based on MAPE patterns (master/slave, coordinated control, hierarchical control, regional planning and information sharing) proposed by D. Weyns et al. [3] using a visual notation based on our UML profile.

A. Structural features

In the structural modeling step, we propose a UML profile that extends the UML 2.0 component meta-model.

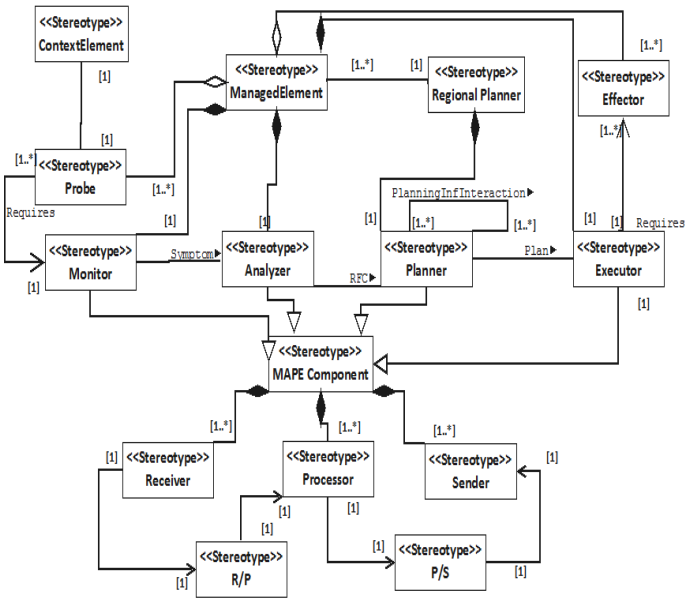

Fig. 2. UML profile of the regional planning pattern: conceptual model

In this paper, we particularly present the instantiation of the regional planning pattern. The conceptual model of the regional planning pattern is shown in Figure.2.

The main entities that constitute the architecture of a regional planning pattern are the $\ll Regionalplanner \gg$ and the $\ll ManagedElement \gg$. They represent a sub-class of UML package meta-model. A regional planner has one Planner component. The managed element comprises the application logic that provides the system domain functionality. It contains effector and probe components to perceive and affect the environment. For each region, the Monitor component of the Managed Element monitor the status of the system and possibly the execution environment, the local Analyzer component analyzes the collected information, and reports the analysis results to the associated regional planner.

Regional planners can interact with one another to coordinate adaptions that span multiple regions. Consequently, we model an association $\ll PlanningInfInteratcion \gg$ that represents the interactions between different regional planners.

We noticed that each MAPE component contains three subcomponents, namely a receiver, a processor and a sender.

- The $\ll Sender \gg$ sub-component publishes symptoms, request for change (RFC), plans, etc.
- The $\ll Processor \gg$ sub-component aggregates and filters events, analyzes data, processes plan, etc.
- The $\ll Receiver \gg$ sub-component receives information and monitoring data.

The $\ll Receiver \gg$ and $\ll Processor \gg$ are connected through the $\ll R/P \gg$ dependency. The $\ll Processor \gg$ and $\ll Sender \gg$ are connected through the $\ll P/S \gg$

dependency.

In the first level, we create a very abstract model composed of two regional planners supervising two regions. Each region contains monitor, analyzer and executor components. The regional planner contains one planner component. In this step we concentrate on the communication between the different MAPE components. In the second level, we add the different MAPE sub-components (sender, processor and receiver) and their connections. In the final level, we introduce other entities such as the different POP, effector components, etc.. and their different connections.
Due to space constraint we present only the final modeling level as shown in Fig.3

B. Behavioral features

We provide a modeling solution for describing step-by-step behavioral features of self-adaptive systems using a UML activity-based custom profile.
In the first level , we create a very abstract model composed of two regional planners and two managed elements.
Then, in the second level, we add the different MAPE sub-activities like collect symptom, interpret measurement, etc.
Finally, we add activities related to probes and effectors components as shown in Fig.4. The probe components provide an activity for measuring meteorological data such as humidity, temperature and gas. These measurements are then collected and filtered by the monitor phase of the managed element. Then, the CCC analyzes and interprets them. The CCC sends a request for change to its regional planner if there is a possible fire detection. Consequently, the regional planner plans several adaptation actions to extinguish fire as soon as possible. To achieve this, each CCC notifies effectors to execute adaptation actions such as sending a UAV or vehicle.
UML lacks a formal semantics and does not allow the formal verification of self-adaptive system properties especially temporal properties for critical systems. Consequently, we propose the transformation of a self-adaptive system model that instantiates a MAPE design pattern to Event-B specifications. The major rules allowing the transformation of a graphical model into an Event-B specification is proposed in [5] [6]. Then, the generated specifications are imported into the Rodin theorem prover tool supporting Event-B.

IV. Time patterns for Self-adaptive Systems

In this section, we present the different patterns which are related to temporal properties. They are classified into three categories. We identified 6 different time patterns. We divide these into 3 distinct categories based on their semantics from [7]. The first category: Time lags contains a catalog of time patterns describing a time pattern for expressing time lags between the different MAPE events. The second category Restricting execution times contains a catalog of time patterns for specifying time constraints regarding possible execution times of a single MAPE event or a specific action in the system. The third category comprises recurrent patterns for expressing temporal constraints in connection with recurrent events. We define, next, the three categories of time patterns.

A. Time lags

The first pattern category we consider comprises one pattern expressing the time lags between the different events.

1) Time lags between events:

- Description: This pattern allows defining time lags between two events; i.e., to express a minimum or maximum temporal distance between them.
- Utility: When the analysis module of the CCC detects fire at time t, five to ten seconds latter, the planning module of the regional supervisor must operate.
  To model the time lags constraint, we add a guard in the event planning. Let Ta be the occurrence time of the event analysis. Let $maxT$ be the maximum time lags between the event analysis and the event planning. Let $minT$ be the minimum time lags between the event analysis and the event planning. Also, we add the event Tick-Tock which progress time (Act1) in an Event-B model. The specification of the time lags constraint is presented as follows:

EVENT $Analysis$	**EVENT** $Tick - Tock$
WHERE	**THEN**
$Grd1 : G(A)$	$Act1 : time := time + 1$
THEN	**END**
$Act1 : Acts(A)$	
$Act2 : Ta := time$	
END	
EVENT $Planning$	
WHERE	
$Grd1 : G(P)$	
$Grd2 : Ta + minT < Time < Ta + maxT$	
THEN	
$Act1 : Acts(P)$	
END	

B. Recurrent Patterns

An activity or an event can be executed many times according to a temporal constraint. In this category, there are three patterns:

1) Periodicity:

- Description: An event must be executed every T period of time.
- Utility: For instance, the temperature sensor must sense temperature every 100 s.
  To model the periodicity constraint, we add to the event monitoring a guard in order to respect the periodicity constraint. The specification of the periodicity constraint is presented as follows:

EVENT $Monitoring$	**EVENT** $Tick - Tock$
WHERE	**THEN**
$Grd1 : G(E)$	$Act1 : time := time + 1$
$Grd2 : Time = Tm + P$	**END**
THEN	
$Act1 : Acts(E)$	
END	

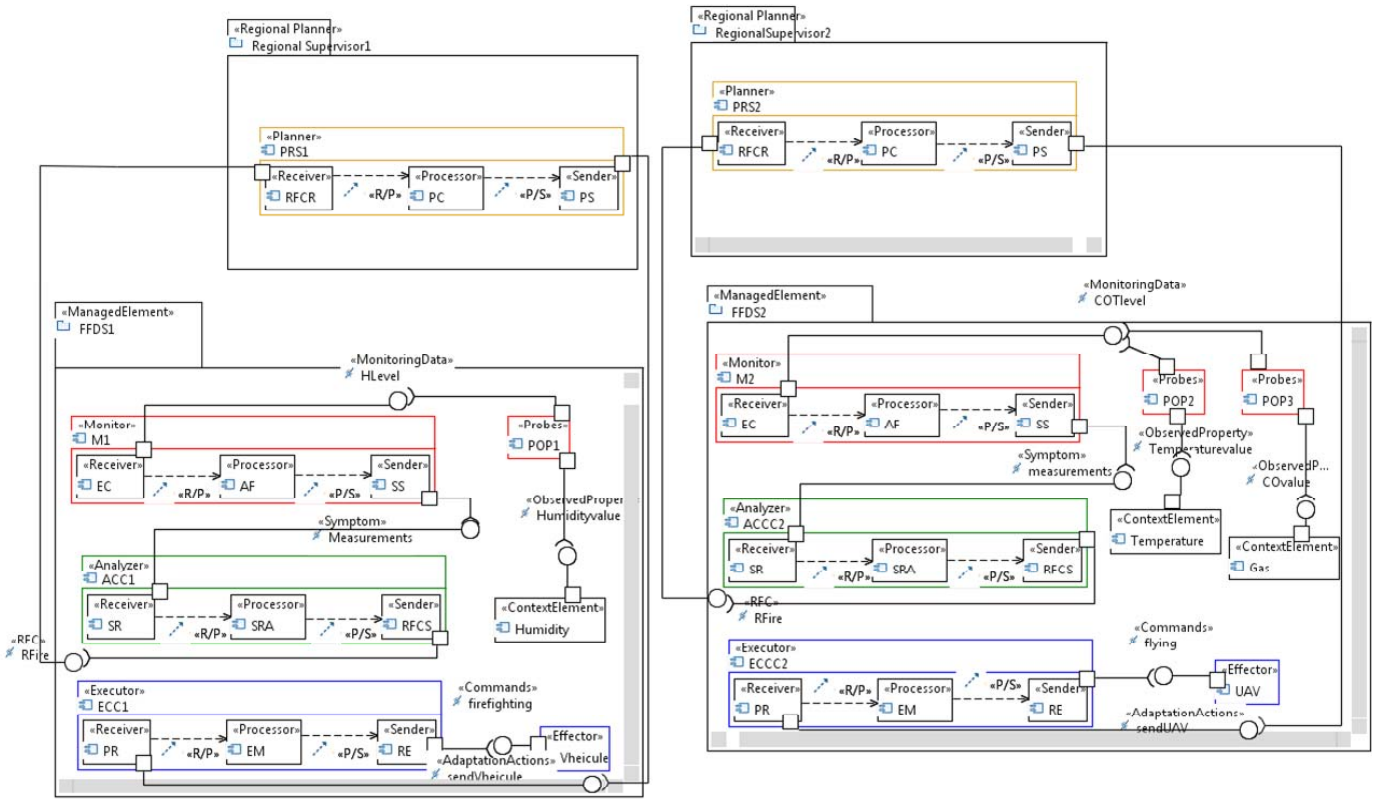

Fig. 3. Regional Planning pattern applied to the FFDS use case: Level-2 (structural modeling).

2) Stabilization time:

- Description: It defines the time lags between two succeeding iterations of the MAPE control loop.
- Utility: When the executor module of the CCC executes an adaptation action, it is labeled unstable. After a reconfiguration time ($ExDuration$), a timer with a certain stabilization time (ST) is set. When this time expires, the unstable state will change to the stable state and a new adaptation control loop can begin with the monitoring event.
  Lets Te be the starting time of the event Execute. Lets ST be the time required for the system to stabilize. Lets $ExDuration$ be the required time for executing an adaptation action. The stabilization time constraint is defined in the event Execute as: $Time > Te + ExDuration + ST$.
  The stabilization time property is specified as shown below.

EVENT $Monitoring$
WHERE
$Grd1 : Time > Te+ ExDuration + ST$
THEN
$Act1 : Acts(M)$
$Act2 : Tm := time$
END

EVENT $Execute$
WHERE
$Grd1 : G(E)$
THEN
$Act1 : Acts(E)$
$Act2 : Te := Time$
END

EVENT $Tick - Tock$
THEN
$Act1 : time := time + 1$
END

C. Restricting execution times

The second pattern category comprises time patterns for restricting the execution times of an event such as specifying the earliest start or latest completion time of an event.

1) Temporal constraint over cardinality:

- Description: It is used to restrict the number of times a particular event or action in a Self-adaptive system may be executed within a predefined time frame.
- Utility: A given adaptation action such as sending UAV can be executed successively at most N times within a time period T.
  There are two steps in order to add a TCOC constraint in our Event-B specification. First, the number of times of the execution of a given adaptation action is recorded in a variable $Nbex$ (Act2). Then, in the event execute a guard (Grd2) is needed which forces the event to be eligible to occur only if the number of times $Nbex$ is less than N and the current time is less than the time period T.
  The TCOC property is specified as follows:

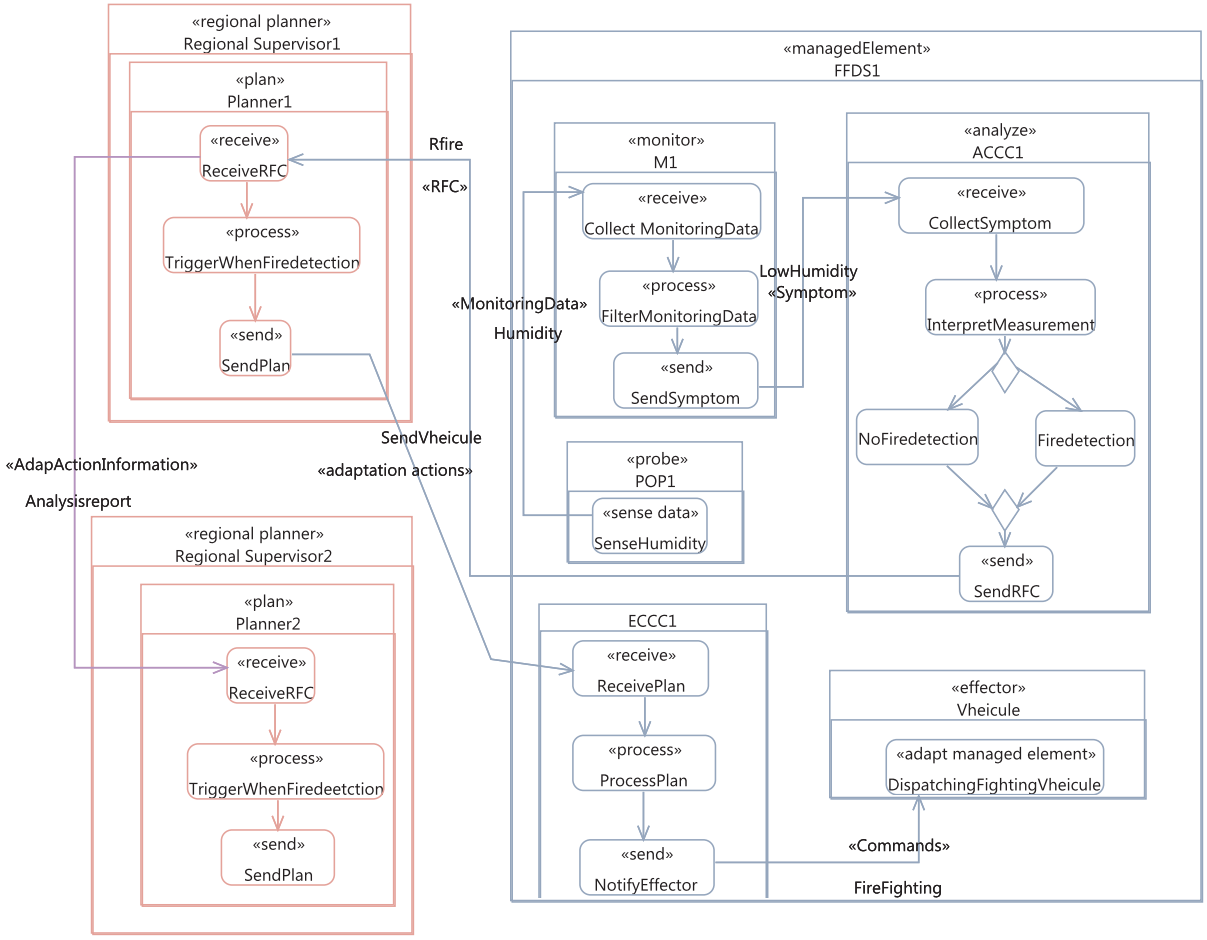

Fig. 4. An exerpt of the regional planning pattern applied to the FFDS use case: Level-2 (behavioral modeling).

EVENT $Execute$	**EVENT** $Tick - Tock$
WHERE	**THEN**
$Grd1 : G(E)$	$Act1 : time := time + 1$
$Grd2 : Nbex < N \wedge time < T$	**END**
THEN	
$Act1 : Acts(E)$	
$Act2 : Nbex := Nbex + 1$	
END	

2) Expiry time:

- Description: A given event can not occur if the expiry period has been passed. An Event B cannot occur after time E of event A occurring.
- Utility: If the planning module of the regional supervisor receives a RFC from the analysis module of the CCC which the expiry time was achieved, this request will not be considered.

  In order to verify expiry time constraint of the event Planning, an action (Act1) is needed to record the occurrence time Ta in the trigger event (event Analysis), and a guard (Grd2) in the restricted event (event Planning) to prevent it from happening if the expiry period E has been passed.

The specification of this property for the event Planning is presented as follows:

EVENT $Analysis$	**EVENT** $Tick - Tock$
WHERE	**THEN**
$Grd1 : G(A)$	$Act1 : time := time + 1$
THEN	**END**
$Act1 : Acts(A)$	
$Act2 : Ta := time$	
END	
EVENT $Planning$	
WHERE	
$Grd1 : G(P)$	
$Grd2 : Time < Ta + E$	
THEN	
$Act1 : Acts(P)$	
END	

3) Delay time:

- Description: An event cannot occur within time W of event A occurring. A given occurred event in a Self-adaptive system must start after the execution of the previous event to avoid overlapping between different events.

- Utility: The event Analysis in the CCC can not occur before the completion of the event Monitoring.
  Lets Tm be the starting time of the monitoring event. Lets $Time$ be the current time of the analysis event and lets $MDuration$ be the duration of the event monitoring. The Delay time constraint is defined in the event analysis as: $Time > Tm + MDuration$. There are two steps in order to add a delay constraint to an event-b model. First the occurrence time of the monitoring event is recorded in a variable Tm. Then, in the event analysis which should be delayed, a guard (Grd2) is needed which forces the event to be eligible to occur after the stated delay period D has been passed from the occurrence of the trigger event. A general pattern of the two events (monitoring and analysis) with the delay property is presented as follows:

EVENT $Monitoring$	**EVENT** $Tick - Tock$
WHERE	**THEN**
$Grd1 : G(M)$	$Act1 : time := time + 1$
THEN	**END**
$Act1 : Acts(M)$	
$Act2 : Tm := time$	
END	
EVENT $Analysis$	
WHERE	
$Grd1 : G(A)$	
$Grd2 : Time > Tm + MDuration$	
THEN	
$Act1 : Acts(A)$	
END	

V. Related Work

The approach presented in this paper has been mainly influenced by different related work on formal specification and verification of self-adaptive systems.

In [8], a concrete UML-based CSML (concern specific modeling language) named ACML (Adapt Case Modeling Language) is used to model structural and behavioral concerns of self-adaptive software systems. These models are then formally defined and taken as input for a model checker. The model checker's result is used to create a quality analysis report (absence of deadlocks, stability, etc.) that is provided to the system designer during design-time.

The authors in [9] propose modeling and analyzing MAPE-K loop for self-adaptation using the concept of multi-agent ASM (Abstract State Machines). They have proposed a verification technique of different properties (flexibility and robustness) through model checking, to discover unwanted interference between MAPE loops.

M. Camilli et al. [10] propose a formal approach to specify and verify the self-adaptive behavior of real-time systems. Their specification formalism is based on Time-Basic Petri nets, a particular timed extension of Petri nets. They propose verification of inter-zone properties to check the correctness of the behavior of the entire system. Also they proposed verification of intra-zone properties to check the correctness of the behavior of a single zone of interest such as invariant, safety, liveness and temporal constraints. They use the GRAPHGEN software tool as a simulation technique to verify correctness of properties. Due to the complexity of such systems, simulation and testing alone are no longer sufficient to gain a sufficiently high quality of the design.

Most of the existing related work that propose formal verification of self-adaptive systems used model checking technique. However, this technique is known to be computationally expensive, as it suffers from the state space explosion problem. Furthermore, existing research approaches don't target the verification of temporal constraint or they only propose a verification of a limited set of temporal constraints.

VI. Conclusion and future work

In this paper, we presented a refinement-based approach to model and specify structural and behavioral features for self-adaptive systems. Also, we presented a set of time patterns to capture the time dimension in self-adaptive systems and to check that all events meet deadlines. As a part of our future work, we plan to integrate the back-tracking technique to help expert in the localization of error in case of property violation.

References

[1] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.

[2] D. Weyns, M. U. Iftikhar, D. G. de la Iglesia, and T. Ahmad, "A survey of formal methods in self-adaptive systems," in *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*, ser. C3S2E '12. New York, NY, USA: ACM, 2012, pp. 67–79.

[3] D. Weyns, B. Schmerl, V. Grassi, S. Malek, R. Mirandola, C. Prehofer, J. Wuttke, J. Andersson, H. Giese, and K. M. Goschka, *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, ch. On Patterns for Decentralized Control in Self-Adaptive Systems, pp. 76–107.

[4] J.-R. Abrial, *Modeling in Event-B: System and Software Engineering*, 1st ed. New York, NY, USA: Cambridge University Press, 2010.

[5] M. Hachicha, E. Dammak, R. B. Halima, and A. H. Kacem, "A correct by construction approach for modeling and formalizing self-adaptive systems," in *2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, May 2016, pp. 379–384.

[6] M. Hachicha, R. B. Halima, and A. H. Kacem, "Modeling and verifying self-adaptive systems: A refinement approach," in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Oct 2016, pp. 003 967–003 972.

[7] A. Lanz, B. Weber, and M. Reichert, "Time patterns for process-aware information systems," *Requirements Engineering*, vol. 19, no. 2, pp. 113–141, 2014. [Online]. Available: http://dx.doi.org/10.1007/s00766-012-0162-3

[8] M. Luckey and G. Engels, "High-quality specification of self-adaptive software systems," in *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 143–152.

[9] P. Arcaini, E. Riccobene, and P. Scandurra, "Modeling and analyzing mape-k feedback loops for self-adaptation," in *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 13–23.

[10] M. Camilli, A. Gargantini, and P. Scandurra, "Specifying and verifying real-time self-adaptive systems," in *Software Reliability Engineering (ISSRE), 2015 IEEE 26th International Symposium on*, Nov 2015, pp. 303–313.