Towards a New Adaptation Engine for Self-Adaptation of BPMN Processes Instances

Jamila Oukharijane[1], Imen Ben Said[1], Mohamed Amine Chaabane[1], Eric Andonoff[2] and Rafik Bouaziz[1]
[1]*MIRACL, University of Sfax, Route de l'Aéroport, BP 1088, 3018 Sfax, Tunisia*
[2]*IRIT, University Toulouse 1-Capitole, 2 Rue du Doyen Gabriel Marty, 31042 Toulouse Cedex, France*

Keywords: BPMN, Self-Adaptation, Autonomic Computing, Adaptation Engine, MAPE-K, Context, Version, BPMN4V-Context.

Abstract: In this paper we introduce an adaptation engine supporting self-adaptation of running BPMN process instances. This adaptation engine implements the MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) approach from autonomic computing for self-adaptation. The MAPE control loop aims at identifying the adaptation need and defining and executing the operations required to deal with these needs while the K is the knowledge needed for the MAPE control loop. More precisely, the paper presents the architecture of the adaptation engine: it details how autonomic managers responsible for self-adaptation of process instances implement the MAPE control loop.

1 INTRODUCTION

It is now widely acknowledged that processes are fundamental in companies since they serve as a support for the alignment between the information systems of companies and their business strategies. However, the dynamic environment in which companies are involved forces them to frequently adapt their processes to face these changes. Moreover, to remain competitive it is crucial that companies take these changes into account as quickly and efficiently as possible. Thus the key to success for companies is closely related to their ability to automatically and autonomously capture changes occurring in their environment and adapt their processes accordingly. As it is not possible to anticipate these changes and catch them into process schemas/models at build-time (most of them are unforeseeable: resources unavailability, new customer requirements), it is important to be able to manage them at run-time. In other words, it is important to be able to monitor and manage the run-time of process instances to identify adaptation needs and define and perform the required operations to implement them. As a consequence, the need for *automating process adaptation at run-time* is more and more strong.

Weber and Reichert have defined a typology for classifying process adaptation needs at run-time (Reichert and Weber, 2012). This typology has identified the following adaptation needs for running business process instances: (i) *adaptation by deviation* for handling occasional unforeseen changes or exceptions in process instances at run-time without changing its process model, (ii) *adaptation by variability* for handling different process variants at run-time depending on the current context situation, (iii) *adaptation by evolution* for handling unforeseen changes in process instances at run-time, which require occasional or permanent modifications in its process model and (iv) *adaptation by looseness* for handling at run-time processes whose process model is not known or incompletely known at design-time and is under-specified.

In the recent past, process adaptation has been highly investigated and numerous solutions have been recommended to help BPM practitioners in managing adaptation manually at run-time (*e.g.*, (Adams et al., 2006), (Adams et al., 2007), (Zhao and Liu, 2013)). However, the manual adaptation of processes is a costly, time consuming and error prone task (Müller et al., 2004), (Reichert and Weber, 2012), (Sprovieri et al., 2016). Thus the

Oukharijane, J., Ben Said, I., Chaabane, M., Andonoff, E. and Bouaziz, R.
Towards a New Adaptation Engine for Self-Adaptation of BPMN Processes Instances.
DOI: 10.5220/0007626602180225
In *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2019)*, pages 218-225
ISBN: 978-989-758-375-9

issue of self-adaptation of processes has to be investigated.

On the other hand, autonomic computing is a field of computer science aiming at building systems that are able to automatically and autonomously adapt their own structure and behavior in response to changes occurring in their operating environment (Horn, 2001). This concept is known as self-adaptation (De Lemos et al., 2010). To achieve the self-adaptation, several solutions have recommended the IBM's MAPE-K approach (IBM, 2006), which is the de facto reference model to design self-adaptive software in the context of autonomic computing. This approach advocates four functions including (i) *Monitor (M)* for collecting data about the managed system and its environment from sensors, filtering and aggregating them into symptoms, (ii) *Analyze (A)* for analyzing the symptoms to detect if changes are required, (iii) *Plan (P)* for constructing the actions needed to resolve detected changes and (iv) *Execute (E)* for applying the actions required to adapt the behavior of the managed system using effectors. These four functions (MAPE) handle and share *Knowledge (K).*

Several contributions have recommended the use of solutions from autonomic computing in the BPM field for making processes self-adaptable (Ayora et al., 2012), (Oliveira et al., 2012), (Oliveira et al., 2013), (Ferro and Rubira, 2015), (Seiger et al., 2016), (Seiger et al., 2017). These contributions are a step forward for self-adaptation of processes. However they have the following drawbacks. First, they do not address the issue of self-adaptation of processes in a comprehensive and global way within the framework of the four adaptation needs defined in Reichert and Weber's taxonomy (Reichert and Weber, 2012): they only take into account one or two adaptation needs but never all four at the same time. Second, most of the contributions mainly focus on the self-adaptation of the behavioral dimension of running process instances (*i.e.*, process activities and their coordination), the informational dimension (*i.e.*, data required to or produced by activity execution) and/or the organizational dimension (*i.e.*, resources involved in activity execution), but neglect the intentional dimension (defining process goals and process constraints), which is another important dimension that must be taken into account to have a comprehensive view of processes.

To overcome these limitations, we recommend an adaptation engine based on the MAPE-K approach from autonomic computing to manage the running of process instances so that they self-adapt to changes occurring in their operating environment. The adaptation engine includes a set of autonomic managers monitoring process components, namely, the process itself and its activities (*i.e.*, sub-processes or tasks). Each manager implements the MAPE control loop responsible for (i) the identification of adaptation needs and (ii) the definition and the execution of the operations required to the implementation of the identified adaptation. These autonomic managers also handle and share knowledge (K). Knowledge modeling is based on a meta-model that allows processes and their activities to be represented in different ways: the notion of version serves as a support for such a representation. The meta-model also makes it possible to define when to use these versions: the notion of context serves as a support for such a definition.

The remainder of the paper is organized as follows. Section 2 is dedicated to the presentation of the main works addressing self-adaption of processes using the MAPE-K approach of autonomic computing. Section 3 presents our contribution to address the issue of self-adaptation of business process instances. It mainly introduces the recommended adaptation engine and details the MAPE control loop. Finally, Section 4 concludes the paper and gives some directions for future works.

2 RELATED WORK

Self-adaptation of processes has been the focus of several contributions such as (Ayora et al., 2012), (Oliveira et al., 2012), (Oliveira et al., 2013), (Ferro and Rubira, 2015), (Seiger et al., 2016) or (Seiger et al., 2017). Each of these contributions recommends the use of the MAPE-K approach for activity (sub-process or tasks) or process monitoring.

For instance, (Ayora et al., 2012) recommended a solution to manage process variants at design time and self-adapt them at run-time. At design time, the solution makes it possible to model process variability by describing process variants using three models: the *base model* to specify process fragments (*i.e.*, consistent part of processes) shared by all process variants, the *variation model* to specify the replacement fragments that can alternatively replace fragments of the base model, and the *resolution model* to specify the context conditions that define the conditions of use for the replacement fragments. At run-time, the recommended solution monitors fragments and dynamically adapts the base model according to the variation model. Thus it compares the context conditions of monitored fragments with the context events recorded in the context model,

and eventually defines an adaptation plan gathering actions for fragments replacement.

The contribution described in (Oliveira et al., 2012) and (Oliveira et al., 2013) introduced the MABUP (Multi-Level Autonomic Business Process) approach for self-adaptation of processes. MABUP recommends two main steps. The *modeling step* supports the modeling of processes considering four abstraction levels: the organizational level, the technological level, the operational level and the service level. The part relating to self-adaptation is more particularly defined in the technological level, which identifies monitored tasks, and in the operational level, which defines the self-adaptation of monitored tasks in terms of variant, variation point and context. The other step of MABUP is the *management step* that uses the MAPE-K approach to self-adapt processes. This step checks all the variation points and evaluates the context in each variant to identify adaptation needs. If adaptations are required, it selects the variation point and thus the variant that satisfies the context of the operating environment.

Another interesting contribution is (Ferro and Rubira, 2015). This contribution introduced an adaptation engine for self-adaptation of process instances at run-time. The adaptation engine is based on the MAPE-K approach. It introduces three types of agents, the agents *monitor*, *adapter* and *executor* that implement the different steps of the approach. The agent adaptor implements decision making for adaptation, which is driven by goal and business rule analysis. It also implements the operations possibly required for process adaptation, in two steps: first, it selects existing activities in the process repository or it uses planning technique for reconfiguration of activity coordination analyzing pre-conditions, post-conditions, interdependences between activities; second, it uses business rules to define the required operations for process adaptation.

Finally, (Seiger et al., 2016) and (Seiger et al., 2017) suggest the self-adaptation of process instances based on their goals. More precisely, the authors proposed a framework for enabling self-adaptive business process in cyber-physical systems (CPS) based on the MAPE-K approach. The Monitor function collects context elements from the physical world related to task goals. These context elements are then analyzed to check for CPS consistency after task execution with respect to task goals. In case an inconsistency is detected, *i.e.*, a task goal is not satisfied, the task instance is (i) adapted by replacing the resource involved in task execution by another and (ii) executed afterwards in order to try to restore CPS consistency and continue with process execution as planned.

Table 1 evaluates the previous contributions with respect to (Oukharijane et al., 2018) criteria related to self-adaptation of process instances at run-time and giving answers to the following questions:

- What components are monitored?
- What process dimensions are affected by adaptation?
- What is the basis for adaptation decision making?
- Does the approach allow traceability?
- What process adaptation needs are taken into account

The first and main observation we can make from Table 1 is that all the examined contributions only partially consider the adaptation needs of Reichert and Weber's taxonomy (Reichert and Weber, 2012). This is mainly due to the chosen modeling approach for adaptation. Indeed the variant-based approach, recommended for instance

Table 1: Related work evaluation.

Criteria \ Works	(Ayora et al., 2012)	(Oliveira et al., 2012) (Oliveira et al., 2013)	(Ferro and Rubira, 2015)	(Seiger et al., 2016) (Seiger et al., 2017)
monitored components	fragment (task, sub-process or process)	task	task	task
impacted process dimensions	informational organizational behavioral	informational organizational	informational organizational behavioral	organizational
decision-making for adaptation	any process variable	any process variable	goal process variable	goal process variable
adaptation achievement	variant-based	variant-based	goal-based rule-based	goal-based
traceability	not supported	not supported	not supported	not supported
process adaptation needs	variability looseness	variability	deviation looseness	deviation

in (Ayora et al., 2012) and in (Oliveira et al., 2012), (Oliveira et al., 2013), allows the modeling of several variants for fragments or tasks, each variant being convenient to a given context. Only one fragment schema is kept for each variant and there is no track of the different changes performed to each variant (none of these works supports traceability). However, evolution is not related only to the ability to perform updates on process, sub-process or task schemas but also to the ability to keep track of these updates.

Second, we can observe that the process dimensions impacted by the adaptation differ from a contribution to another. For instance, in (Ayora et al., 2012), the adaptation may impact the informational, organizational and behavioral dimensions of process, while in (Seiger et al., 2016) and (Seiger et al., 2017), only the organizational dimension of process may be adapted.

The third observation that can be made from Table 1 relates to decision-making for adaptation and its achievement. In the examined works, decision-making for adaptation is based on the comparison of the values that the process variables have in the operating environment and the variation points or process goals. However, in both cases, operating environment modeling is not addressed in a comprehensive way, taking into account all the process dimensions. On the other hand, adaptation techniques of the examined contributions are either variant-based or goal-based. As explained before, the variant-based approach does not allow dealing with adaptation by evolution. Moreover, we believe that these techniques could be suitably mixed to improve adaptation achievement.

In this paper, we recommend an adaptation engine that ensures the self-adaptation of process instances and addresses the drawbacks of the works previously examined.

First, as in examined contributions, we recommend the MAPE-K approach for self-adaptation. Indeed the use of this approach reduces human involvement when adapting the running process instances. It also ensures a modular separation between the process engine and the adaptation engine in order to overcome the drawbacks of embedding the self-adaptation logic within the process engine. Finally, it makes processes highly reactive to changes occurring in their operating environment.

Second, in contrast with the examined works, our adaptation engine monitors business process instances at the following abstraction levels: (i) the *process* and the *sub-process* levels, to support adaptation by variability and adaptation by evolution; and (ii) the *task* level to support adaptation by deviation (including unavailability of data or resources) and adaptation by looseness (including late binding). Thus we take into account the main adaptation needs for process instances at run-time. Moreover, considering these three abstraction levels makes the self-adaptation of all the dimensions of processes possible.

Third, we recommend versioning as adaptation technique for adaptation achievement. The reasons of this choice are as follows. First, the notion of version has been recognized as a key notion to deal with adaptation by variability, adaptation by evolution, adaptation by deviation and adaptation by looseness (Zhao and Liu, 2013), (Ben Said et al., 2014). Notably, versioning allows supporting both adaptation by variability and adaptation by evolution, as several alternatives (variants) of a same process (or sub-process or task) may be modeled according to the context and also several schema versions can be kept for each variant (traceability is supported in versioning). Second, handling versions of processes facilitates the migration of instances from an initial schema to a final one, allowing, if the migration is not possible, two different instances of the same process (or sub-process) to run according to two different schema versions. Therefore, the adaptation achievement should benefit from the versioning technique.

3 MAPE-K ARCHITECTURE OF THE ADAPTATION ENGINE

This section introduces the architecture of the *Adaptation Engine* (AE) by distinguishing its external architecture, which highlights how it is connected with the outside, from its internal architecture, which highlights how it monitors and possibly adapts process instances at run-time. We also give the life cycles of the components of this internal architecture

3.1 External Architecture

Figure 1 gives an overview of the external architecture of the AE.

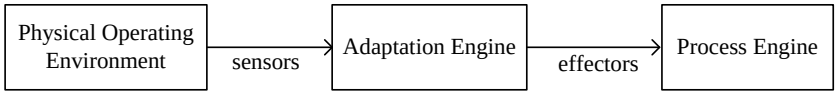

Figure 1: External Architecture of the Adaptation Engine.

The AE is connected to the Physical Operating Environment (POE) in which processes operate and to the Process Engine (PE) that executes them. More precisely it receives data from sensors in the POE, analyses them and eventually sends to the PE (*e.g.*, Camunda, Activiti, Bonita, JBPM…) via effectors the operations to be carried out to adapt process instances. Thus we separate the self-adaptation concern from other concerns. The main advantage of the separation between the AE and the PE is the reusability of the adaptation engine for various process engines. This means that we do not modify the structure of any process engine.

In the same way, we separate the AE from the POE. The main advantage of this separation is that it makes easier the integration of new sensors. Indeed, sensors of the POE are both physical and software entities installed in the physical environment: the physical entity records sensor data while its corresponding computational entity sends to the adaptation engine changes detected in the recorded data. Thus when adding new sensors to the POE, the AE must not be modified since only the POE is impacted (as it integrates both the physical and the computational entities of sensors).

3.2 Internal Architecture

The AE is composed of a set of autonomic managers responsible for the monitoring and the possible adaptation of process instances at run-time. There are as many autonomic managers as there are running process instances. Each of these autonomic managers implements the Monitor, Analyze, Plan and Execute (MAPE) control loop of autonomic computing to support self-adaptation of process instances and handle and share knowledge (K).

Figure 2 gives an overview of the architecture of an autonomic manager. Regarding the K, it is composed of the version repository and the logical operating environment. The *version repository* stores versions of tasks, sub-processes and processes as well as their contexts of use. These versions and their contexts of use are modeled according to the BPMN4V-Context meta-model, which is an extension of BPMN supporting the modeling of versions of BPMN components (*i.e.*, processes, sub-processes, tasks, roles and data), along with their use contexts. As for the *Logical Operating Environment* (LOE), it gives an accurate picture of the running environment of process instances. It is composed of a set of context elements that correspond to data sent by sensors from the POE and that refer to process instance variables from any context category (immediate, internal, external or environmental). These data are only sent by sensors if their values change.

Regarding the MAPE, each step is implemented by separate software components. The *Monitor* component aims at getting an accurate picture of the operating environment of each monitored process instance along with its sub-processes and tasks. It receives data from the POE and records them in the LOE. Each data sent from the POE corresponds to a variable of the monitored process instance. This variable is modeled in the LOE as a context parameter, which can be from any context category, and for which the Monitor component stores a value. Note that this component follows a specific life cycle, linked to the monitoring it must provide on the POE.

Contrary, the three other components of the MAPE control loop, namely A (Analyze), P (Plan) and E (Execute), follow the same life cycle as they respectively implement the identification of the possible need for adaptation, the definition of the operations required to address this need and the execution of these operations in the process engine. Communication between components is visualized by thick arrows in Figure 2. There is no arrow between the M and the APE because the life cycle of these components are independent.

The *Analyze* component implements the *A* of the control loop. It is responsible for the identification of the need for adaptation. It is composed of several *task analyzers* (as many as tasks in the process) that match their use context with the current context, *i.e.*, the context described in the LOE, and identify the need for adaptation if the matching fails. Each task analyzer notifies the analyzer of the component in which it is involved and which is either a sub-process analyzer or a process analyzer. In turn, the sub-process analyzer notifies the analyzer of the process in which it is involved. Note that notifications between analyzers are visualized in Figure 2 as thin arrows.

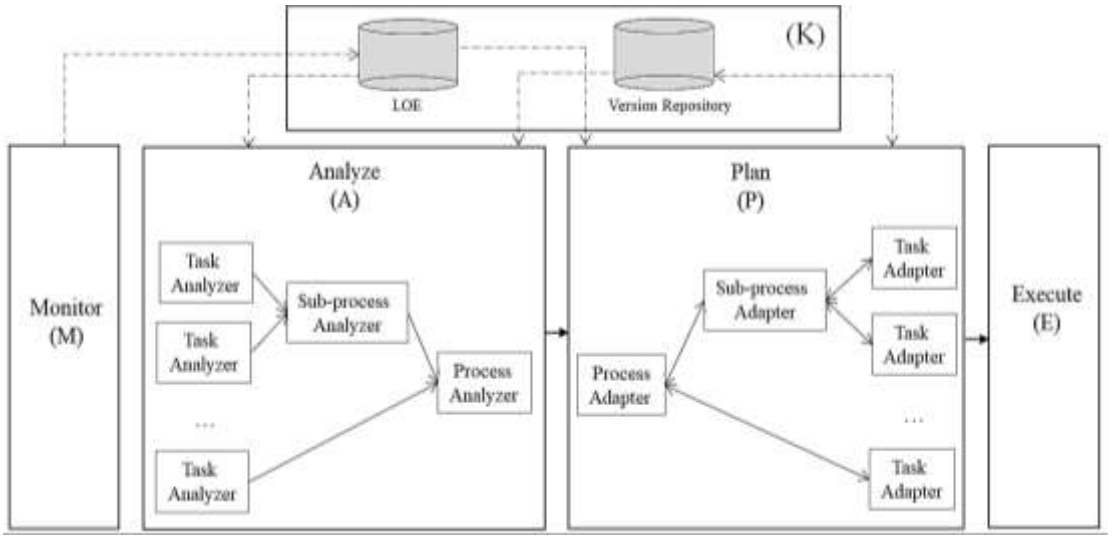

Figure 2: Architecture of an Autonomic Manager.

To sum up, the identification of the need for adaptation is based on task analysis (is their context of use checked in the LOE?) and, if identified, the adaptation need is also forwarded to the sub-process or the process in which the considered task is involved so that these latter are informed that an adaptation is required.

The *Plan* component implements the *P* of the control loop. It receives as an input the different adaptation needs identified by the Analyzer component (several tasks may have identified a need for adaptation) and produces as an output the set of operations to be carried out to meet these adaptation needs. These operations are expressed using the adaptation patterns recommended in (Weber et al., 2008), which correspond to high-level operations making changes on the process instance schema easier to perform. The *Plan* component is composed of several adapters that define needed operations to implement the identified adaptation: one process adapter, as many sub-process adapters as sub-processes in the monitored process instance and as many task adapters as tasks in the monitored process instance. Each adapter deals with its own adaptation and defines the operations required to carry it out. Thus a task adapter deals with adaptation of the monitored task, and mainly addresses *adaptation by looseness* (late biding, unavailability of resources or data), and *adaptation by deviation* (repeat, skip or redo a task). A sub-process adapter and a process adapter deal with adaptation by evolution and adaptation by looseness (late modeling) as they can modify the coordination of theirs tasks (and theirs sub-processes) and insert, delete, move, or replace tasks (and/or sub-processes). In addition, sub-process adapters and process adapters also deal with *adaptation by variability* as they can replace a version of a task, sub-process or process with another one by matching the context of use of the considered versions (defined in the version repository) with the current context (defined in the LOE). However, if it does not exist any existing solution modeled as a version in the version repository, the adapter either requests for domain expert intervention or builds on its own a new solution (process or sub-process) in accordance with the objectives to meet and the current situation, as in (Rantrua et al., 2013). In addition, the *Plan* component is also responsible for updating the version repository if it is possible to specify new versions from defined adaptation operations.

The *Execute* component implements the *E* of the control loop. It receives from the *Plan* component the operations to be carried out and maps these operations into operations that can be understood by the target process engine. Each process engine being specific, we have defined several wrappers suitable for each of them and it is these wrappers who map the operations defined by the *Plan* component into operations of the process engine. In addition, the Execute component can also use specific operations of the process engine that allow suspending, resuming or restarting the execution of a process instance. Ultimately it triggers the execution of these mapped adaptation operations in the target process engine.

Finally, the dotted arrows of Figure 2 visualize the read/write operations in K of the Monitor, Analyze, Plan and Execute components.

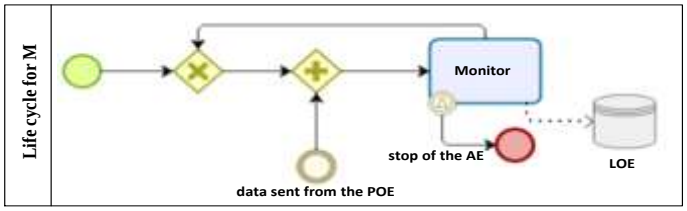

Figure 3: Life cycle for M (Monitor) as a BPMN diagram.

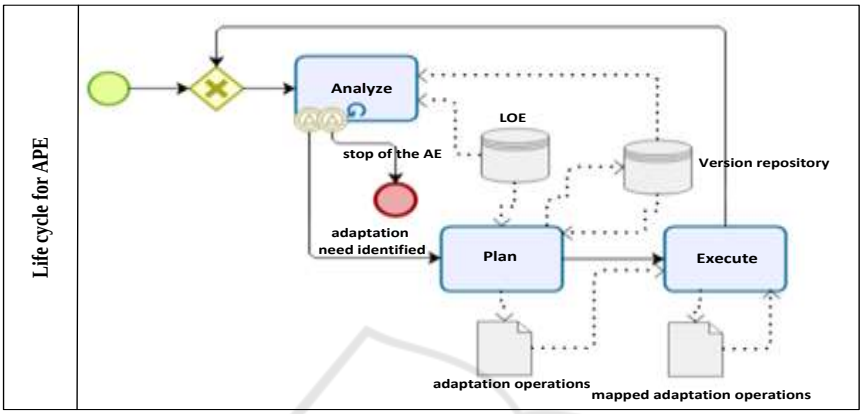

Figure 4: Life cycle for APE (Analyze, Plan and Execute) as a BPMN diagram.

3.3 Life Cycles for MAPE

Figures 3 and 4 respectively give the life cycles of M and APE as BPMN diagrams. Regarding M, its life cycle indicates that the activity Monitor reacts to each data sent from the POE and updates the LOE accordingly. This life cycle ends when the adaptation engine is stopped. Regarding APE, the first task is Analyze, which supports the identification need from data stored in the LOE and in the Version repository. If a need for adaptation is identified then the tasks Plan and Execute are triggered. Plan defines the set of operations to be carried out using the adaptation patterns of (Weber et al., 2008) while Execute maps the adaptation operations according to the target process engine and executes the mapped adaptation operations. Adaptation operations and mapped adaptation operations are modeled in the BPMN diagram as data objects produces and/or consumed by the activities Plan and Execute. This life cycle ends when the adaptation engine is stopped.

4 CONCLUSION

This paper has addressed the issue of process self-adaptation, which is an important issue in the BPM field. To address this issue, the paper has recommended an Adaptation Engine (AE) based on the MAPE-K approach of autonomic computing so that monitored process instances self-adapt to changes occurring in their operating environment. The adaptation engine includes a set of autonomic managers monitoring process components. Each manager implements a MAPE control loop responsible for the identification of adaptation needs and for the definition and execution of the operations required to the implementation of the identified adaptation. These autonomic managers also handle and share knowledge (K).

Paper contributions are as follows. First, the paper has defined the architecture of the AE. It has discussed the interaction between the AE and the operating environment through sensors and between the AE and the process engine through connectors. Second, it has presented the internal architecture of the autonomic managers implementing the MAPE control loop, including their life cycle.

Benefits of our approach are as follows. First the separation between the AE and the process engine with which it interacts makes the AE reusable for various process engines. In the same way, the separation between the AE and the physical operating environment, integrating sensors responsible for the measurement of data describing the operating environment of processes, makes easier the integration of new sensors. Second, the

recommended solution benefits from the MAPE-K approach advantages. This approach from the autonomic computing field makes possible the implementation of self-adaptation for monitored components thanks to autonomic managers implementing MAPE control loop and sharing K(knowledge). Third, the combined use of contexts and versions makes it possible to take into account the different adaptation needs identified in Reichert and Weber's taxonomy. Thus our recommended solution supports in a coherent framework the self-variation, self-looseness, self-deviation and self-evolution of monitored process instances.

However, our contribution is incomplete as the recommended solution has only dealt with the architecture of the AE, notably the internal architecture of the autonomic managers. In our future work, we have planned (i) to implement this architecture using multi-agent systems in which each component of MAPE control loop can be considered as a software agent; (ii) to address the modelling of the knowledge handled and shared by autonomic managers; and (iii) to address the connection with different process engines and notably the specification of adaptation operations and the mapping of these adaptation operations according to the target process engine.

REFERENCES

Adams M. et al. 2007. Dynamic, extensible and context-aware exception handling for workflows. Int. Conference on the Move to Meaningful Internet Systems, Vilamoura, Portugal, pp. 95–112.

Adams M. et al. 2006. Worklets: a service-oriented implementation of dynamic flexibility in workflows. Int. Conference on the Move to Meaningful Internet Systems, Montpellier, France, pp. 291–308.

Ayora C. et al., 2012. Applying CVL to business process variability management. VARiability for You Workshop (@MODELS), Insbruck, Austria, pp. 26–31.

Ben Said I., et al., 2014. Extending BPMN 2.0 Meta-models for Process Version Modelling. Int. Conference on Enterprise Information Systems, Lisbon, Portugal, April 2014, pp. 384–393.

Ferro S., Rubira C., 2015. An architecture for dynamic self-adaptation in workflows. Int. Conference on Software Engineering Research and Practice, Las Vegas, Nevada, USA.

Horn P., 2001. Autonomic computing: IBM's perspective on the state of information technology. https://www.researchgate.net/publication/200031805_Autonomic_Computing_IBM's_Perspective_on_the_State_of_Information_Technology

IBM, 2006. An architectural blueprint for autonomic computing. *IBM White Paper*.

De Lemos R. et al., 2010. Software engineering for self-adaptive systems: a second research roadmap. Int. seminar on Software Engineering for Self-Adaptive Systems, Dagstuhl Castle, Germany, pp. 1–32.

Müller R., et al. 2004. Agentwork: a workflow system supporting rule-based workflow adaptation. *Data and Knowledge Engineering*, 51(2), pp.223–256.

Oliveira K., et al. 2012. A multi-level approach to autonomic business process. Brazilian Symposium on Software Engineering, Natal, Brazil, pp. 91–100.

Oliveira K., et al. 2013. Multi-level autonomic business process management. Int. Conference on Enterprise, Business-Process and Information Systems Modeling, Valencia, Spain, pp. 184–198.

Oukharijane J. et al., 2018. A survey of Self-Adaptive Business Processes. Int. Business Information Management Association Conference, Seville, Spain, pp. 1388–1403.

Rantrua A., et al. 2013. Flexible and emergent workflow using adaptive agents. Int. Conference on Computational Collective Intelligence, Craiova, Romania, pp. 185–194.

Reichert M., Weber B., 2012. Enabling flexibility in process-aware information systems: challenges, methods, technologies. Springer.

Seiger R., et al., 2016. Enabling self-adaptive workflow for Cyber-physical Systems. Int. Conference on Enterprise, Business-Process and Information Systems Modeling, Ljubljana, Slovenia, pp. 3–17.

Seiger R. et al. 2017. Toward a framework for self-adaptive workflows in cyber-physical systems. *Software & Systems Modeling*. Springer, pp. 1–18.

Sprovieri D., et al. 2016. Run-time planning of case-based business processes. Int. Conference on Research Challenges in Information Science, Grenoble, France pp. 1–6.

Weber B., et al 2008. Change patterns and change support features - enhancing flexibility in process-aware information systems. *Data and Knowledge Engineering*, 66(3), pp. 438–466.

Zhao X., Liu C., 2013. Version management for business process schema evolution. *Information Systems*, 38(8), pp. 1046–1069.