Component Ensemble-based UML/MARTE Extensions for the Design of Dynamic Cyber-Physical Systems

Nissaf Fredj[1], Yessine Hadj Kacem[1], Olfa Kanoun[2] and Mohamed Abid[1]
[1]*CES Laboratory, ENIS, University of Sfax, Tunisia*
[2]*Chair of Measurement and Sensor Technology, Chemnitz University of Technology, Germany*

Keywords: Open-ended CPS, MARTE, Component Ensemble, Closed-loop.

Abstract: Cyber-Physical Systems (CPS) are open-ended distributed systems which incorporate autonomous components that interact with clear responsibilities to fulfill a defined objective. They are subject to several issues related to the interaction between components and dynamic behavior of their status as well as real-time constraints. Their design requires effective modeling formalisms for functional verification and real-time analysis of CPS at early stages of development. Unified Modeling languages and especially the Modeling Analysis of Real-Time and Embedded systems (MARTE) profile fail to capture the dynamics of CPS. They focus on individual components and are unable to provide such QoS guarantees. Moreover, they again assume static component architectures, which effectively hinder their direct application in open-ended systems. In this respect, we proposed to extend MARTE profile with the concept of ensemble semantics for the design of a dynamic grouping of CPS components. The objective is to save resources consumption such as battery life and allow for scalable QoS guaranties. Moreover, the present approach proposes designs patterns for the specification of CPS closed-loop concerns which addresses the components composition, adaptation as well as the verification of system constraints.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

1 INTRODUCTION

Cyber-physical systems consist of a large number of devices that monitor and react to their environment and interact with each other through input-output interfaces. They are complex and subject to numerous constraints due to their environment mobility and resources fluctuation (Bradai et al., 2020). As a result, there is a necessity to evaluate system output quality at an early stage of development to obtain reliable systems which permit an efficient and fast reaction to the environment fluctuations. Lightening the task of CPS designers and raising the abstraction level have become prominent solutions to cope with their complexity and avoid executions problems. To do so, several research studies tried to exploit mainly the modeling languages such as the UML/MARTE profile and component-based methods to specify and develop open-ended dynamic CPS. However, standard techniques (Fredj et al., 2020) do not address the dynamic and cooperative aspects of CPS components. Such works which deal with high-level specifications of adaptation of real-time and embedded systems (FREDJ et al., 2018), do not support the interaction between dynamic components which is essential in open-ended applications. On the other hand, component ensemble-based works (Masrur et al., 2017) (Crnkovic et al., 2011), which allow modeling large-scale dynamic systems by a set of interacting components, do not provide a basis for specifying the adaptation and runtime requirements. Besides, they fail to capture the dynamic CPS at a high-level of abstraction and provide reusable and generic solutions. To address the above issues, we propose an extension in MARTE profile to support the concepts of component ensemble which are required to specify the interaction between dynamic components of CPS. This innovative extension enhance MARTE profile to support the design of dynamic open-ended CPS at a high-level of abstraction. Moreover, the present work aims at specifying the closed-loop concerns for dynamic systems based on proposed design patterns as generic and reusable solutions to enhance the separation of concerns and improve system modularity. The proposed patterns cope with the composition between components, monitoring and adaptation logic which are responsible for specifying dynamic open-ended CPS.

The present proposal is evaluated through an Intelligent Crossroad System (ICS). Section 2 surveys the related works about the high-level design and component ensemble-based approaches for the specification

Fredj, N., Kacem, Y., Kanoun, O. and Abid, M.
Component Ensemble-based UML/MARTE Extensions for the Design of Dynamic Cyber-Physical Systems.
DOI: 10.5220/0010546401580166
In *Proceedings of the 16th International Conference on Software Technologies (ICSOFT 2021)*, pages 158-166
ISBN: 978-989-758-523-4
Copyright © 2021 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

of dynamic open-ended CPS and the evaluation of system performance. Section 3 presents the proposed extensions and the closed-loop patterns. In section 4, an illustration of the proposal through the ICS example is described. Finally, section 5 concludes this work.

2 RELATED WORKS

We classify the state-of-the-art studies about the high-level methods, which emerged in the context of dynamic Cyber Physical Systems (CPS) design. In particular, we discuss their support for cooperative and adaptive architecture as well as real-time analysis.

2.1 Component Ensemble-based Approaches for Developing Dynamic Open-ended Systems

Most of works in (Bures et al., 2013) (Crnkovic et al., 2011) were built based on the concept of component ensemble. They define an ensemble as the semantic interaction between components and constitute their composition. The DEECo (Crnkovic et al., 2011) framework allows modeling closed-loop-based dynamic systems and provides mechanisms to describe interactions between them. The components dynamically join an ensemble at runtime, which describes the components on the basis of their roles, as one component has a coordinator role and other are members. They consist of knowledge which reflects its current state and have a set of processes that manipulate their knowledge. Then, the dynamic interaction between the coordinator and members consists in exchanging attribute values according to knowledge description. Thereby, the work of (Masrur et al., 2017) proposed a design methodology based on the concept of ensemble and focus on the autonomous behavior, adaptation and components collaboration. In fact, they extended the DEECO framework (Crnkovic et al., 2011) to guarantee real-time behavior and constraints analysis. Previous works (Crnkovic et al., 2011) (Bures et al., 2013) (Masrur et al., 2017) have allowed reasoning about real-time requirements and provided deterministic semantics for real-time analysis and distributed collaboration of CPS. However, they present a set of challenges. First, DEECO closed loop concerns are either mixed with the system functional logic, thus add complexity to the knowledge adaptation process and lucks reusability. Besides, the adaptation of components knowledge and real-time analysis were supported at a low-level and dependent on the target system. Finally, previous component-based designs methods have failed to capture the dynamic CPS at a high-level of abstraction. They do not provide reusable and generic solutions which are independent from any system. Therefore, developing such complex systems using approaches that handle technical details at low-level, is no more an efficient solution.

2.2 High-level-based Approaches for the Design of Dynamic Real-time Systems

Recently, there is a clear trend to use high-level models for the design and development of complex dynamics systems. Several research studies adopted the system abstraction as an optimal way to improve adaptive and distributed systems design (Beux, 2007) by using model-based methods and UML profiles. An UML extension has been implemented in the form of MAFRTE profile, which is often the most adapted to the design of real-time and embedded systems (Fredj et al., 2020). A model-based process for dynamic reconfiguration in distributed real-time systems was proposed in (Krichen et al.,). The solution is based on a non-predefined set of configurations specified by new extensions in MARTE and meta-models. The dynamic reconfigurable system is modeled by a set of modes each of which is associated with a configuration described by a set of components and connections between them. In (Corsaro et al., 2002), the author proposed a virtual component pattern, which permits the adaptation of a distributed application to the embedded systems' memory constraints. The work in (IGLESIA and WEYNS, 2015) proposed a formal specification-based template of distributed self-adaptive mobile learning application which makes the system robust to the GPS accuracy degrading. The self-adaptation is based on a MAPE (Monitor, Analyzer, Planner, and Executer) control loop for distributed mobile applications. The weakness of this work is that the verification template is specific to the functionality of the proposed use case. In (Arcaini et al., 2015), the authors presented a framework to formally model and verify the distributed self-adaptive systems using the concept of multi-agent ASM (Abstract State Machines). Their work supports techniques for the validation and verification of adaptation scenarios, as well as the correctness of the adaptation logic. The authors in (FREDJ et al., 2018) proposed an extension in MARTE profile to design the runtime behavior of adaptive real-time systems. It offers a well-structured support to extend the UML/MARTE profile with runtime and

reconfiguration semantics and nonfunctional properties. This work was extended in (Fredj et al., 2020) where the proposed extensions were integrated in a model-based development process for the analysis of adaptive real-time system behavior and system constraints (energy consumption and task deadlines). Unfortunately, MARTE extensions-based works (FREDJ et al., 2018) (Fredj et al., 2020) assume closed component architectures, which effectively prevent their application in case of dynamic CPS development. Furthermore, previous works fail to target open-ended CPS requirements where components interact one another and their architecture changes continuously at runtime.

We derived from the state-of-the-art studies the weakness of the existing methods for the design and the analysis of dynamic cyber-physical systems. Most of studied approaches (Crnkovic et al., 2011) (Masrur et al., 2017) proposed component-based solutions to address cyber-physical applications requirements and support dynamic interaction between components. These solutions are not abstract enough and fail to address adaptation and communication requirements as well as real-time analysis at an abstract level for an early verification of the system. In this respect, the approaches of (FREDJ et al., 2018) (Arcaini et al., 2015) (IGLESIA and WEYNS, 2015) (Fredj et al., 2020) provided promising solutions to raise the abstraction level and cope with the complexity of low-level adaptation tools. Unfortunately, they proposed solutions which largely fail to address the requirements of distributed adaptive systems. This is mostly because they are unable to model an open-ended system, in which cooperative components interact with one another to adapt to the current state of its environment.

3 OUR PROPOSAL

We derived from the state-of-the-art studies the weakness of the existing methods and tools for the design and the analysis of dynamic cyber-physical systems. They do not operate at a high-level of abstraction and provide sufficient means to model interracting components behavior, adaptation logic, and real-time constraints. To address the above issue, we propose to extend MARTE profile by the concepts of ensemble to support the interaction between system components and allow reasoning about dynamic systems behavior and runtime semantic of CPS. Moreover, we propose two design patterns as generic and reusable solutions for the specification of DEECO closed-loop concerns for the interaction between dynamic components of CPS. This helps to separate both the adaptation logic and components composition features from functional system logic. The first design pattern focuses on monitoring of components adaptation changes. The second pattern describes the composition between components based on the concept of ensemble. and the assessment of closed-loop parameters (such as ensemble evaluation and computation delays) based on a set of metrics. The patterns are independent from target low-level tools to enhance the separation of concerns and improve system modularity. More precisely, they focused on the adaptation behavior and interaction of systems components as well as the analysis of real-time constrains.

In what follows, we will present the proposed extensions in MARTE profile for designing dynamic and adaptive CPS. Afterwards, we will introduce the specification of the closed-loop based design patterns. Finally, we will illustrate the proposed extensions and closed-loop patterns through ICS case study.

3.1 Component Ensemble Extensions in MARTE Profile for the Design of CPS

The proposed extension was introduced as a new package which extends the Software Resource Modeling (SRM) package of MARTE and uses stereotypes from the MARTE:RuntimeContext, MARTE:Timed Element, Causality package and MARTE:Communication package. More precisely, we need four sub-profiles of MARTE. First, since we are interested in cyber-physical systems composed of adaptive components which are managed by real-time tasks named process, we use concepts from the SRM (Software Resource Modeling) package of MARTE. SRM is a specialization of the Generic Resource Modeling (GRM) package to describe the structure of the system which is composed of software multitasking. Second, in order to follow the system behavior changes at runtime, we need a behavioral modeling and runtime semantics concepts provided respectively by the CommonBehavior as well as the RuntimeContext sub-packages. Finally, we need to use the timing concepts offered by the TimedElement package to define the components resource execution period. In addition, we use some concepts from Communication package to support requests between adaptive components.

Figure 1 illustrates the proposed package, in which we have distinguished the standard stereotypes of MARTE from the proposed ones by the green color.

≪Ensemble≫ stereotype:

It inherits implementation and attributes of RtUnit

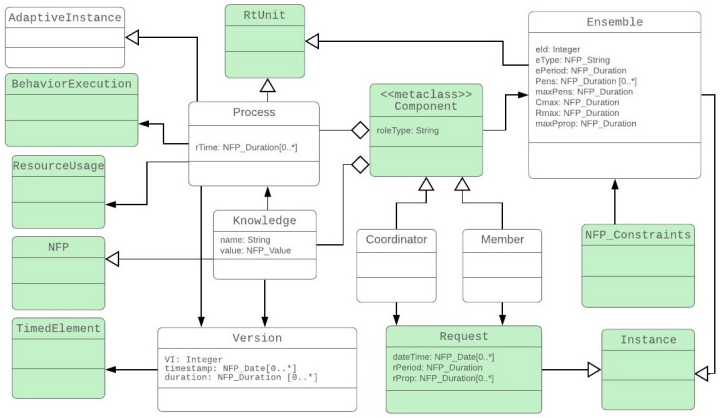

Figure 1: Component ensemble-based extensions in UML/MARTE profile.

from HLAM sub-package of MARTE. RtUnit stereotype specifies active and schedulable resources.

- Generalization: 1) MARTE::RtUnit. 2) MARTE::Instance stereotype: A system can establish a set of ensemble instances at runtime. Thus, the Ensemble stereotype inherits from the MARTE::Instance stereotype which specifies runtime instances.
- Associations: 1) UML Component meta-class: Ensemble is a composition of system components thus, it is associated to the UML Component meta-class. 2) MARTE::NFP_Constraint: Ensemble is associated to NFP_Constraint stereotype of MARTE that is a boolean condition to determine whether two components should form an ensemble.
- Attributes: 1) eId: identify the ensemble. 2) ePeriod: period of an ensemble 3) eType: type of ensemble. 4) Pens: delay of ensemble instance evaluation. 5) maxPens: max delay of an ensemble instance evaluation. 6) Cmax: sum of computation delay of all processes in an ensemble. 7) Rmax: sum of reaction time delay of all processes in an ensemble. 8) maxPprop: max delay of knowledge propagation in an ensemble.
- Semantics: An ensemble defines a semantic connector between components and constitutes their composition. It can be considered as real-time task which is externalized from the component itself.

≪Coordinator≫ ≪Member≫ stereotypes: They inherit the implementation and attributes of UML::Component meta-class.

- Generalization: UML::Component meta-class
- Associations: Request: communication between coordinator and member is based on a set of requests. Thus ≪ Coordinator≫ and ≪Member≫ stereotypes are associated to the Request stereotype of MARTE::Communication package.
- Semantics: When specifying an ensemble, prospective components are described by roles. One component in the ensemble has a coordinator role, whereas the remaining components are members of the ensemble.

≪ Knowledge ≫ stereotypes: It inherits the attribut of MARTE::NFP stereotype.

- Generalization: MARTE::NFP
- Associations: 1) UML::Component: System component is specified by a set of knowledge. 2) Version stereotype: Each knowledge has a set of versions which should be tracked and stored for future adaptation decisions. Knowledge stereotype is associated to Version stereotype which was proposed in (FREDJ et al., 2018). A Version stereotype has a VI (version identifier) attribute and timestamp and duration attributes which have a multiplicity of [0..*]. Version stereotype is associated with the TimedDurationObservation and TimedInstantObservation stereotypes of TimedElement package to define the lifetime and instant of each version respectively (FREDJ et al., 2018).
- Semantics: Knowledge is measurable data that characterize components. It is a state variable that may change their values at runtime. Knowledge can be a QoS attribute or an NFP property (such as CPU utilization, memory usage, power consumption, network bandwidth).

≪ Process≫ Stereotype: It inherits the implementation and attributes of RtUnit stereotype, which specifies active and schedulable objects (resources). Process is an adaptive object that inherits from AdaptiveInstance stereotype. The latter was defined in (FREDJ

et al., 2018) to extend MARTE profile for supporting the specification of adaptive resource at runtime.

- Generalization: 1) MARTE::RtUnit. 2) MARTE::AdaptiveInstance.
- Associations: 1) MARTE::ResourceUsage: Process stereotype is associated to the ResourceUsage stereotype of MARTE which define the amount of consumed resource. 2) Knowledge: Knowledge values represent inputs to process. 3) UML::Component: System component is composed of a set of process. 4) MARTE::BehaviorExecution: Processes have runtime behaviors specified by the BehaviorExecution stereotype of MARTE::RuntimeContext package.
- Attributes: 1) WCRT: worst case response time of a process. 2) rTime: response time of a process.
- Semantics: Process is a real-time and adaptive unit, which responsible to manage component knowledge values.

≪ Request≫ Stereotype: It inherits implementation and attributes of MARTE::Instance stereotype of MARTE.

- Generalization: MARTE:: Instance
- Associations: 1) Coordinator: Coordinator component sends request to members. 2) Member: Member component receives request from coordinator.
- Attributes: 1) dateTime: timestamp of a request instance. 2) rPeriod: period of a request. 3) rProp: delay of a request instance propagation.
- Semantics: Request is a standard stereotype of MARTE which represents an instance of communication and specifies data that are passed between coordinator and member. We define new attribute rProp with [0..*] multiplicities to denote that a request may has several instances in a period of time, each of which has a propagation time (rProp) value.

The system modeling based mainly on the proposed extensions have to go through a model-based development process for the automatic generation of the dynamic CPS from a high-level specification as well as the evaluation of the real-time constraints.

3.2 Design Patterns for the Specification of DEECo Closed-loop

The second contributions of this work is to separately specify the behavior of closed-loop using design patterns. In fact, it should be noted that for most of existing approaches, the adaptation logic as well as the components membership evaluation are either mixed with the system functional logic. To overcome this problem, we propose two design patterns which interact one another to perform a component ensemble evaluation and knowledge changes monitoring and adaptation.

3.2.1 Ensemble Evaluator Pattern

The Ensemble evaluator pattern is based on the concept of ensemble. It constitutes a generic solution that can be applied to any distributed dynamic system. In fact, it is required to determine whether two components (a coordinator and a member) should form an ensemble. This pattern is responsible for checking the membership condition between components which is expressed as a logic predicate formulated upon coordinators and member's knowledge. The exchanging of attribute values between coordinator and member is performed according to the description given in the knowledge exchange specification. Additionally, this pattern evaluates the closed-loop performance (ensemble evaluation delay, reaction time and propagation time) each unit of time by calculating metrics. The description of the proposed pattern follows the template in (Buschmann et al., 1996) which presents a set of rubrics to describe a design pattern such as problem, context, intent, and solution.

- Problem: Evaluator pattern deals with the issue of distributed systems components composition and the issue of closed-loop cost-effectiveness. It treats the question of with whom to interact and how well interact do.
- Context: This pattern is used when there is a need of components composition to exchange knowledge between distributed systems components.
- Intent: This pattern is responsible for checking the membership condition between two components to form an ensemble of and to evaluate the closed-loop performance.
- Solution
  Structural view: The structural view of a design pattern is represented by a class diagram in Figure 2 which contains: 1) Evaluator class represents a real-time task, which identifies components roles and evaluate their relationship based on a logical condition. It takes as input component knowledge attributes values and generates as output an ensemble exchange description. Evaluator class is annotated by the Ensemble stereotype from the proposed package in Figure 1. It assesses closed-loop performance and adjusts such control parameters to ameliorate the closed-loop cost-effectiveness. 2) ControlParameter represents a

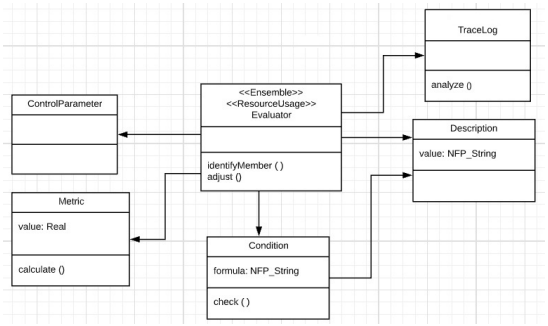

Figure 2: Ensemble pattern structure.

closed-loop parameter which can be adjusted to new context variations. For instance, it can be the ensemble evaluation period or thresholds. 3) Condition class represents a logical expression to check membership between components. Condition satisfaction is managed by evaluator. 4) Description class represents the specification of knowledge attributes exchange which is generated after membership condition checking. 5) Metric class specifies measurement which are established by Evaluator to assess closed-loop performance. 6) TraceLog is a repository that serves to store knowledge values changes and ensemble evaluation metrics Stored data are represented by traces. Each trace is associated with a version identifier and has a specific value at a specific timestamp.

Behavioral view: A behavioral view illustrates the pattern behavior through the invocations of methods between objetcs. The Evaluator starts by checking the membership condition (method check ()) satisfaction. Once condition is cheeked, Evaluator recovers the Boolean value. If the membership condition is satisfied, Evaluator identifies the role of the considered component as a member. These steps are repeated each unit of time during ensemble period (ePeriod). The Evaluator calculates close-loop performance metrics (methods calculate ()) and analyzes the history (TraceLog) to make statistics. Building on these calculations, and via its adjust () method, it generate a decision to adjust the closed-loop performance by acting on some ControlParameters.

3.2.2 Adaptation Pattern

In the present work, the specification of adaptation logic is performed using design pattern as generic solution to increase the productivity and reusability.

- Problem: The adaptation pattern deals with the issue of dynamic adaptation as a reccurent problem (IGLESIA and WEYNS, 2015). It treats the question of what and how to adapt?

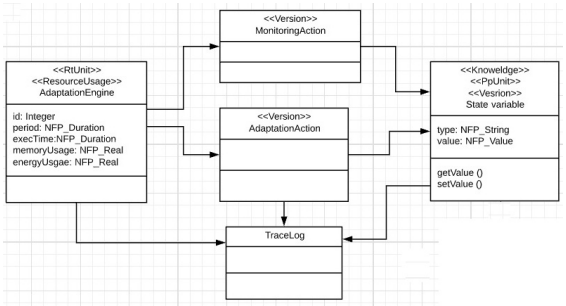

Figure 3: Adaptation pattern structure.

- Context: This pattern is used when a state variable is changed and attached to an adaptive process.
- Intent: This pattern is responsible for monitoring and adaptation of the component knowledge values and storing adaptation traces in the history.
- Solution
  Structural view : The static view of adaptation pattern is illustrated in Figure 3 which contains: 1) AdaptationEngine class represents a real-time unit that observes the state variable status and decides if a variation has occurred or a threshold (Threshold class) has been exceeded. It adapts state variable values in order to meet system constraints. It is specified by the RtUnit and ResourceUsage stereotypes of MARTE. 2) StateVariable class represents component knowledge that may change their values under different instants. Thus, they are annotated by the proposed Version stereotype. Thanks to VI attribute of Version stereotype, designers are able to load the required version from the history. 3) ChangeableElement class represents the component that has to be adapted. 4) AdaptationAction class represents adaptation actions which are applied by the adaptation engine to the changeable element in the system. 5) The TraceLog is a repository of adaptation decisions previously generated by the adaptation engine. It may serve to store the state variables changes which are represented by traces.

Behavioral view: The monitoring action begins with sensors which periodically conveys a new status measure of the supervised state variable then it notifies the adaptation engine. The AdaptationEngine receives the new measure and updates the state variable with the new status value and stores old value in the history (TraceLog). Afterwards, it inspects the new status to decide about the change relevance. It can use thresholds to prove whether the measure is in the interval delimited by min and max values. The negative case indicates an irregular state causing the AdaptationEngine to generate an adaptation action and update the changeable element.

4 ICS EXAMPLE ILLUSTRATION

In order to illustrate the proposed extensions, we rely on an Intelligent Crossroad System (ICS). More precisely, the illustration focuses on the specification of input model of the considered system based on the proposed extensions and the closed-loop patterns. In fact, the architecture of the ICS system is composed of two components/nodes which are the car and the ICS. Vehicles arrive and leave the intersection region where their priorities changes at different instants according to their distances to the center of the intersection. Each node is occupied by a processing unit. The system enables to concurrently run different real-time tasks or process. A process can be executed in response to a timer event or as a reaction to a change in one of its attributes.

4.0.1 ICS System Modeling

Figure 4 illustrates the class diagram of ICS system model. It is composed of two classes Car and ICS to specify the car and ICS nodes respectively. Car and ICS classes are annotated by Member and Coordinator stereotypes, respectively, from the proposed package in Figure 1. Car component is characterized by a set of knowledge; speed, mode, distance which represent the speed of car, assignment speed mode and distance from the crosscutting, respectively and the time of the last speed update, which must be less than a threshold. Each attribute is annotated by the knowledge stereotype from the proposed package. Car component runs three process which are the monitorSpeed, updateSpeed and calculateSpeed. The monitor process focus on monitoring whether the time of the last speed update must be lower than a threshold. The calculate process is responsible for computing the speeds of potential cars depending on the current traffic situation and the time at which they can reach the intersection. The update process adjusts the cars speeds to avoid conflicts.

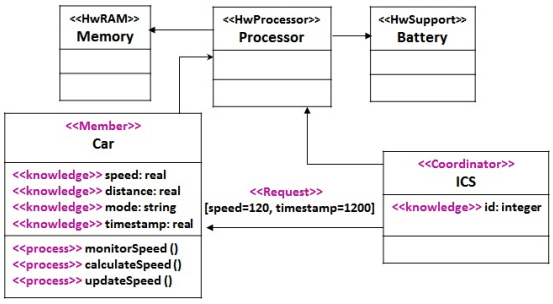

Figure 4: ICS system model.

Component/node processes are annotated by Process stereotype. They are executed in processing unit which is specified by Processor class and annotated by MARTE::HwProcessor stereotype. Battery and Memory classes are annotated by MARTE::HwSupport and MARTE::HwMemory stereotypes, respectively.

4.0.2 Closed-loop Patterns Application to the ICS Example

Ensemble pattern starts by checking ensemble condition which must be evaluated to true at both the ICS and the car nodes, separately. This checking requires knowledge values input data such as car speed, distance and current mode. Thus, ensemble pattern is integrated in the ICS input model of Figure 4 by two associations between its Evaluator class (see Figure 2) and Car and ICS classes of ICS model (see Figure 4). The integration of the closed loop patterns in the ICS model is illustrated in the resulting model of Figure 5. The Evaluator is responsible for checking whether input data is obsolete, if not it stores new values in the history (TraceLog class). We assume that membership condition is related to the distance of car to the crosscutting which must be lower than 60 m. If the condition is validated, Evaluator identifies cars which satisfy the condition, as members to dynamically join the ensemble. Then, Evaluator generates as output knowledge description which needs to be exchanged from the car to the ICS and from ICS to car. Once data are received by the car component, Evaluator establishes a set of metrics to determinate the Pens at this timestamp and compares it to maxPens. The adaptation of car speed is separately managed by the proposed adaptation pattern depending on the description generated by ensemble pattern. Thus, ensemble and adaptation patterns are combined together by adding a new association between Description class of ensemble pattern and AdaptationEngine class of adaptation pattern (see Figure 2 and Figure 3). In fact, car speeds represent the state variable which are captured and monitored using monitoring actions (MonitoringAction Class) of the adaptation pattern. Thus, an association between the Car class of the ICS model and the MonitoringAction class of the adaptation pattern is established (see Figure 5). Afterwards, the adaptation engine examines the collected values and generates adaptation actions which are applied to the changeable element represented by the updateSpeed() process. Thus, the adaptation pattern is combined with the ICS model by a new association between its AdaptationAction class and the changeable element represented by the updateSpeed process of the Car class (see Figure 5). Adaptation actions are stored each timestamp in the history

(TaceLog) for future adaptation decision. Once the speed adaptation is performed, the ensemble pattern is invoked to recover the updateSpeed process reaction time. Then, Evaluator performs a set of metrics to assess the closed-loop delay which is computed based on Pens and rProp values. In fact, once a membership evaluation is performed at the ICS, a car propagates its knowledge to the ICS immediately and the data is received at the ICS Pens time later. The knowledge is propagated with a rProp delay from the car to the ICS. Similarly, the ICS propagates its knowledge to the corresponding car just after a membership evaluation is performed and data is received Pens time later at the car. The ICS's knowledge is propagated to the car rProp time later. The attributes rProp and Pens are from Request and Ensemble stereotypes 1, respectively, denote the knowledge propagation and the ensemble evaluation period of a node i in the ICS system.

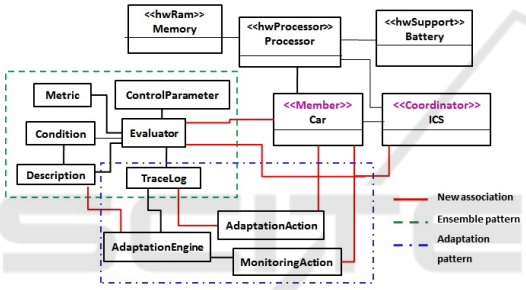

Figure 5: Application of Closed loop patterns to ICS system.

5 CONCLUSION AND DISCUSSION

The use of high-level methods for the specification and the analysis of cyber-physical systems provide advantages upon low-level development. It consists in raising the level of abstraction to describe cooperative dynamic components systems. Almost existing approaches do not support the high-level design for monitoring, adaptation and real-time analysis concerns. They do not operate at an abstract level and do not support the traceability of dynamic components versions during execution. To overcome these issues, we defined a new component ensemble-based UML/MARTE package to support the dynamic interaction between distributed components and to specify CPS and real-time constraints. Additionally, we have proposed design patterns for specifying closed-loop features. In fact, a pattern view screens out specific details and makes it possible to describe and conceive solutions independently from any system. It offers the advantage of concerns separation to reduce execution problems.

As future works, we plan to integrate the present UML/MARTE extensions and proposed patterns into a complete model driven-based approach for the analysis of distributed systems.

REFERENCES

Arcaini, P., Riccobene, E., and Scandurra, P. (2015). Modeling and analyzing mape-k feedback loops for self adaptation. *International Symposium on Software Engineering for Adaptive and Self Managing Systems, SEAMS15, IEEE, Piscataway, NJ, USA*, page 13–23.

Beux, S. L. (2007). Un flot de conception pour applications de traitement du signal systematique implementees sur fpga a base d ingenierie dirigee par les modeles. *Universite des Sciences et Technologie de Lille*.

Bradai, S., Bouattour, G., Naifar, S., and Kanoun, O. (2020). Electromagnetic energy harvester for battery-free iot solutions. *IEEE World Forum on Internet of Things (WF-IoT)*.

Bures, T., Nicola, R. D., Gerostathopoulos, I., Hoch, N., Kit, M., Koch, N., Monreale, G. V., Montanari, U., Pugliese, R., Serbedzija, N., Wirsing, M., and Zambonelli, F. (2013). A life cycle for the development of autonomic systems: The e mobility showcase. *2013 IEEE 7th International Conference on Self-Adaptation and Self-Organizing Systems Workshops*, pages 359–381.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. (1996). *Pattern Oriented Software Architecture*.

Corsaro, A., Schmidt, D. C., Klefstad, R., and ORyan, C. (2002). Virtual component - a design pattern for memory-constrained embedded applications. In *in Proceedings of the Ninth Conference on Pattern Language of Programs (PLoP*.

Crnkovic, I., Sentilles, S., Vulgarakis, A., and Chaudron, M. R. (2011). A classification framework for software component models. *IEEE Transaction. Software. Engineering. volume 37(5)*, pages 593–615.

Fredj, N., Kacem, Y. H., , and Abid, M. (Dec 2018). Runtime uml marte extensions for the design of adaptive rte systems. *International Conference on Intelligent Systems Design and Applications (ISDA),2018, Vellore India*.

Fredj, N., Kacem, Y. H., and Abid, M. (2020). Runtime model-based framework for specifying and verifying adaptive rte systems. *International Journal of Computer Applications Technologies*.

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, Volume 10, Issue 3*, pages 1–15.

Krichen, F., Hamid, B., Zalila, B., Jmaiel, M., and Coulette, B. Development of reconfigurable distributed embedded systems with a model driven approach. *Con-*

currency and Computation: Practice and Experience, Volume 27, Issue 6.

Masrur, A., Kit, M., Matena, V., Buresb, T., and Hardt, W. (2017). Component-based design of cyber-physical applications with safety-critical requirements. *Journal of Microprocessors and Microsystems volume 10(3)*, page 1–17.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS