Design Pattern for self-adaptive RTE systems monitoring

***Abstract*—Approaches for the development of self-adaptive real-time embedded (RTE) systems are numerous. However, there is still a lack of generic and reusable design which fits different systems and alleviate the designer task. Design patterns represent a promising solution to get fast and reusable design. Unfortunately, patterns dealing with self-adaptive RTE systems development are still not well tackled in the literature. The general structure of self-adaptive RTE systems is based on a MAPE loop which is composed of four basic adaptation processes: Monitor, Analyze, Plan, and Execute. In this paper, we define patterns for the monitoring and analyzing processes through the generalization of relevant existing adaptation approaches to improve their accessibility to new adaptive systems developers. To evaluate the work, the proposed patterns are applied to a relevant existing cross-layer adaptation framework.**

***Keywords*-Self-adaptive system; real-time embedded system; design patterns**

I. Introduction

The development of self-adaptive embedded systems is a costly and time-consuming process due to their complexity, continuously variable environment as well as the lack of reusable designs and development tools [1, 2, 3]. Therefore, there is a high demand for alleviating the designer task and reducing cost and Time To Market by decreasing system complexity and automating the process management. At this moment, a question needs to be answered: How to exploit software engineering approaches to provide solutions that rapidly transform an ordinary embedded system into a self-adaptive one?

Many approaches for the development of self-adaptive RTE systems have been proposed in the literature [4, 3]. They are based on software engineering approaches, typically the Model Driven Engineering (MDE) which is well appropriate to embedded systems and permits to ease their design by avoiding dealing with technical details. However, there is still a lack of reusable design that is target independent and sufficiently generic to fit different systems and fasten the designer task. To address this problem, we propose to use design patterns which are effective for design reusability and quickness [5]. Patterns that are particularly intended for the embedded systems domain are still under-explored. Existing solutions are mostly dedicated to desktop and distributed systems and ignore important constraints of RTE systems, essentially the real-time constraint.

Our aim is to propose patterns for the development of self-adaptive RTE systems. These patterns provide a generic specification that guides experts of the domain to build their own adaptive systems giving them the freedom to insert additional information that have not been defined in the design. The general structure of self-adaptive RTE systems is based on a MAPE loop which is composed of four basic adaptation processes: Monitor, Analyze, Plan, and Execute. In this paper, we propose patterns for the monitoring and analyzing processes. We survey relevant research works related to adaptation in order to construct a unified terminology of adaptive systems that we use to develop our design patterns. These patterns are to be integrated in a model-based approach for automatic pattern-based re-factoring of the system design model to generate a dynamically adaptive system model. The patterns are described using Unified Modeling Language (UML) models that are annotated with the UML/MARTE (Modeling and Analysis of Real-Time and Embedded Systems) [6] profile stereotypes. The MARTE profile offers a rich terminology for the specification and analysis of RTE systems.

The present paper is organized as follows. Section 2 presents the adaptive embedded systems domain and introduces the adaptation loop modules. In section 3, we give the description of the proposed patterns. We illustrate the utilization and effectiveness of the patterns through a case study of dynamically adaptive RTE system running object tracking application. Section 5 provides a brief discussion of related work. To conclude, in Section 6, the suggested patterns are briefly outlined and future work is given.

II. Adaptation loop for RTE systems

A self-adaptive system is a system that is able to change its structure or behavior at run-time in response to the execution context variations and according to adaptation engine decisions [7]. The design of adaptive embedded systems presents many challenges due to the complexity of the problem it handles. A common basic challenge is optimizing system non-functional properties (e.g. maximizing output quality) while meeting internal and external constraints (e.g. real-time constraint). For example, a high quality of service may require a high utilization of system resources, such as CPU cycles and memory space, and implies high energy consumption. Self-adaptation can be conceived in different ways depending on various aspects such as target platform, application domain, adaptation goals, users' requirements, system constraints, context changes, adaptation mechanisms, targeted system layers, adaptation scope, and many others [8]. However, there is a common structure of the adaptation mechanism that a self-adaptive

system embodies. It is an adaptation loop referred to as the MAPE loop [9]. It is composed of sensors, effectors and four basic modules which are the monitoring, analyzing, planning or deciding and executing or acting. These entities are briefly described as follows:

- Sensors collect data about the status of the system and its environment.
- The monitoring module processes the collected data to decide about relevant changes and then trigger change events
- The analyzing process examines the received events to detect if an adaptation is required. It can also identify the source of the change. Monitoring and Analyzing processes stand for all forms of observation and evaluation of systems' execution such as performance monitoring, safety inspection and constraint verification [7].
- The planning process generates an adaptation decision which specifies what elements to change and how to change them in order to best meet system requirements. Two common approaches are used in the literature to construct Decision makers: rule-based approaches and intelligent approaches. The second approach does not fit the real-time and embedded systems domain because of its requirements in terms of computing time. Adaptation actions can be classified in two categories [10]: Parameter adaptation/tuning and compositional adaptation mechanisms. The former modifies application parameters that determine the behavior. The latter exchanges algorithmic or structural system components with others to improve the system outcome.
- The executing module applies the decision to the system. It maps actions to effectors' interfaces.
- An effector is related to an adaptable system element and is responsible for applying adaptation actions to it.

Figure 1 illustrates the structure of a loop-based self-adaptive system. The modules forming this structure are presented as design patterns in order to permit their reuse, separately, in other contexts.

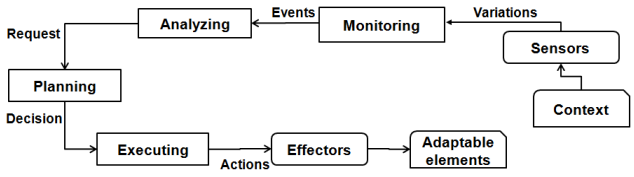

Figure 1. Structure of a loop-based adaptive system

III. Patterns description

As we have already mentioned, in this paper we focus on the two first processes of the adaptation loop; monitoring and analyzing. The proposed patterns follow the [11] pattern template. In this paper, we give details of six fundamental fields which are the pattern name, problem, intent, context, motivation and solution. To describe our patterns, we use standard UML diagrams annotated with the UML/MARTE profile stereotypes. We explain for each pattern both structural and behavioral views using the class and sequence diagrams, respectively. These diagrams are the result of the abstractions and generalization of many relevant adaptation-related works [12, 13, 14, 15, 16, 17, 18, 19, 20, 8].

A. RTE Monitor pattern

Name: RTE Monitor

Problem:

The problem treated by this pattern is the detection of irregular status of an RTE system which results from relevant variations of internal and external context elements.

Intent:

The RTE Monitor pattern permits the continuous control of the status of one or more RTE system properties in order to detect relevant changes and trigger events. It takes into consideration the system stability issue by minimizing events trigger through the selection of only important context variations. It also handles concurrency and real-time features relative to the control operations.

Context:

This pattern is used in the first step of development of a self-adaptive RTE system. The designer has to define the system to adapt and his adaptation requirements by answering the «what to monitor?» question.

Motivation :

The starting point that triggers adaptation mechanism in an RTE system is the context variation detection. Therefore, to be self-adaptive, an RTE system first needs to integrate a monitoring module that permits to continuously control and update the status of its execution context. Additionally, the execution context of an RTE system is very fluctuant so that context variation detection risks being very frequent. Therefore, in order to have a stable adaptive system with the minimum of reconfigurations, a monitoring step is required to restrict the number of treated changes by approving only relevant ones.

Solution:

This pattern represents a monitoring process that permits the observation of status of RTE system context properties.

Structural view:

Figure 2 shows the class diagram that explains the structural view of the pattern.

Participants:

- *ContextElement* represents an internal or external property of the system which is observed by the Monitor, such as CPU load, battery life and network bandwidth. It is a passive unit that carries information about the status of a system property and is concurrently accessed by the Sensor and Monitor. It is thus stereotyped «PpUnit». It

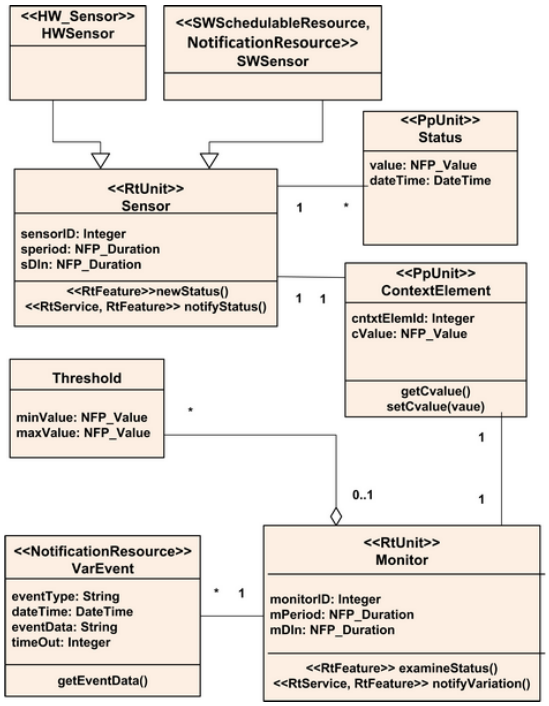

Figure 2. Structural view of the RTE Monitor pattern

specifies its concurrency policy through the *concPolicy* attribute (sequential, guarded or concurrent).

- *Sensors* are responsible for data collection about the status of *Context Elements*. A *Sensor* is associated with each *ContextElement* and provides measures of its status. We classify sensors into two categories: a hardware sensor, stereotyped «HW_sensor», represents a hardware device providing a measure of a physical quantity and converting it into a signal. A software sensor is defined by a software task running concurrently on the system to measure a system property, such as CPU usage, and notify a *Monitor*. It is thus stereotyped «SwSchedulableResource» and «NotificationResource». A notification resource is a software synchronization resource used to notify events. To keep events history, the notified occurrences can be memorized in a buffer by setting the policy attribute to Memorized.
  Since we are in the context of real-time and embedded systems, two basic issues have to be taken into consideration: the concurrency and the real-time features. In order to handle these issues, we annotate the active classes by the HLAM «RtUnit» stereotype[6] indicating that it is a real time unit. An RtUnit is an autonomous execution resource that may own one or more schedulable resources and one or several behaviors. It is also capable of handling different incoming messages at the same time without worrying about concurrency issues thanks to its own concurrency and behavior controller. It owns a message queue permitting to save messages it receives. Messages can represent operation calls, signal occurrences or data receptions. A message can be used to trigger the execution of a behavior owned by the real-time unit. A sensor is an active class that we annotate «RtUnit».
- *Status* stores the measures realized by sensors in order to keep track of context information history which is important to determine the trends of context elements variations [19] and consequently to improve predictions. Status indicates for each measure the date and time and the value. This latter is typed *NFPValue* which has different attributes that permit to precisely specify NFP values, such as statistical qualifier, precision and source.
- The *Monitor* is an «RtUnit» that is associated to each *ContextElement*. It examines sensing data using minimum and maximum values stored in a *Threshold* to decide if a significant variation has occurred or a certain threshold has been exceeded. Threshold may represent either interval limits indicating a regular status or allowed variation margins to be used to decide about the variation relevance. If a variation is relevant, the *Monitor* generates a variation event, stereotyped «NotificationResource».

For the sake of system stability when self-adapting, it is recommended to define an adaptation period in order to manage the adaptation mechanism occurrence. Commonly, this period is equal to a defined number *Ne* of application iterations [16]. Every period, the monitoring process starts a control session and then the adaptation cycle is executed. Therefore, the operations executed by the sensing and monitoring processes need to specify their occurrence kind (such as periodic, aperiodic and sporadic). Moreover, in order to respect the real-time constraint, these operations have deadlines that they are asked to meet. In order to model these real-time properties, we annotate the sensing and monitoring methods with the «RtFeature» stereotype which has the occKind, relDl and absDl (for occurrence kind, relative deadline and absolute deadline, respectively) attributes. In order to specify additional attributes for real-time constraints of these operations, we use the «RtService» stereotype. It permits to manage the execution priority of a real-time service by the specification of the execution kind (exeKind attribute) which can be either deferred, remoteImmediate or localImmediate.

Behavioral view:

Figure 3 shows the UML sequence diagram presenting the execution scenario of the RTE Monitor pattern by showing the communication between the different objects forming it. The monitoring process starts by *Sensor* which periodically delivers a new measure of the status of the supervised *Context Element* then it notifies the *Monitor*. The *NotifyStatus()* method execution kind is *localImmediate*

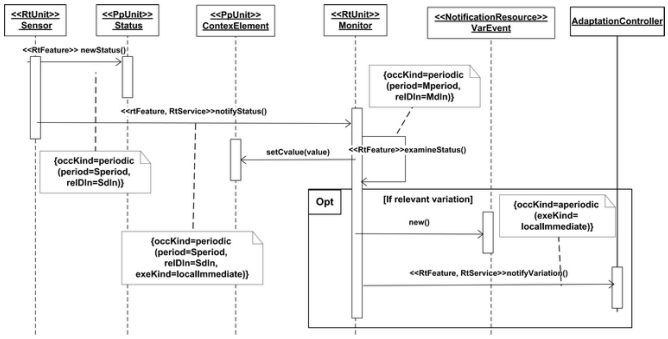

Figure 3. Behavioral view of the RTE Monitor pattern

in order to be immediately executed by the *Sensor*. The *Monitor* receives the new measure and updates the current status value of the *Context Element*. Then it examines the new status to decide about the relevance of the change. It can use thresholds to verify whether the measure is in the interval delimited by min and max values. The negative case indicates an irregular state causing the *Monitor* to generate a variation event and send it to the *Adaptation Controller* of the system through its *notifyVariation()* method in order to be processed and decided upon. This method occurrence is aperiodic since its execution depends on the verification result of the *examineStatus()* method. However, when executed, it has the highest priority, thus having its execution kind set to *localImmediate*.

B. RTE Analyzer pattern

Name: RTE Analyzer

Problem: having the status of an RTE system context, the *RTE Analyzer* pattern responds to the question «Does an adaptation need to be applied?»

Intent:

This pattern permits the verification of constraints meeting of an RTE system and then asks for adaptation if needed. It contributes to providing a stable adaptive system by minimizing adaptation requests. It handles concurrency and real-time features relative to the control operations.

Context:

This pattern is used when designing a self-adaptive RTE system, specifically when information about changes in the system context is available and system constraints are defined.

Motivation:

A change in the execution context does not necessarily affect the functioning of the system, i.e. violate system constraints, thus not requiring an adaptation. Therefore, a verification step is needed in order to avoid useless adaptations.

Solution:

Structural view :

Figure 4 shows the class diagram relative to the structural view of the RTE Analyzer pattern.

Participants:

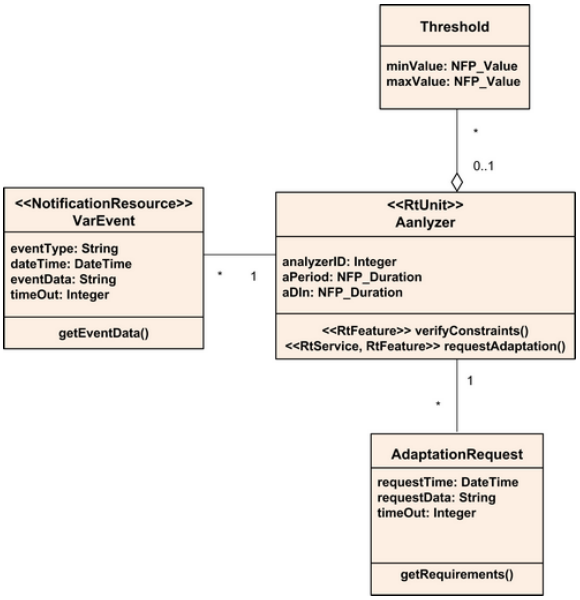

Figure 4. Structural view of the RTE Analyzer pattern

- The *Analyzer* is responsible for the verification of the system constraints meeting. It processes a *Variation Event* that occurs to the system to decide whether an adaptation action is required or not. It is thus an active class stereotyped «RtUnit». It has an analysis method, *verifyConstraints()*, which generally executes a constraint miss test. The miss-test may require *thresholds*. The *requireAdaptation()* method generates an *Adaptation Request*.
- An *AdaptationRequest* carries request data indicating the analysis results such as the source of constraint violation. It has a timeout to be considered when treated.

Behavioral view:

The behavior of the Analyzer pattern is depicted by the UML sequence diagram in Figure 5. Having variation events received in its message queue, the Analyzer treats them in a loop. It asks for event data if the event is still valid, i.e. its timeout is not achieved. Then it uses the collected data to verify the system constraints meeting through the *verifyConstraints()* method. Since events occurrence is aperiodic, this method's occurrence kind is aperiodic too. If constraints are not met, the analyzer asks for adaptation by sending an *AdaptationRequest* to the *Adatation Controller* of the system. For more clarity, we can cite an example of real scenario: when a task entry event occurs in the system, the Analyzer performs a schedulability test to verify the real-time constraint meeting. If tasks' deadlines are not met, it asks for adaptation by generating an adaptation request

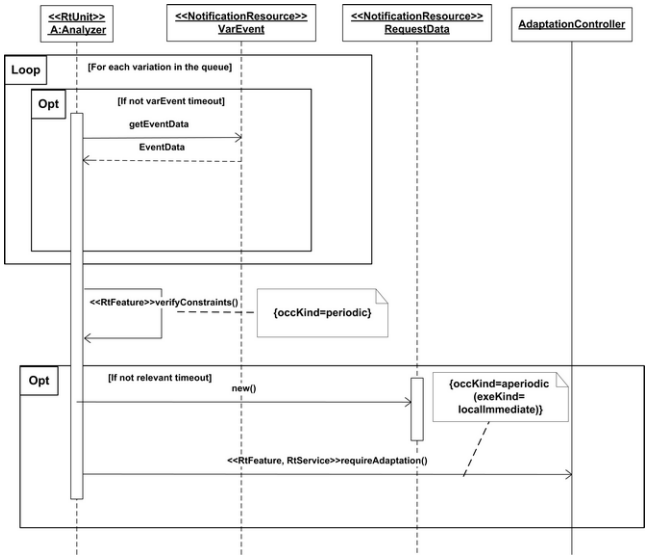

Figure 5. Behavioral view of the RTE Analyzer pattern

carrying new context data.

IV. Example of pattern application

This section permits to illusrate the utilization of the proposed patterns through a case study of a dynamically adaptive object tracking application proposed in [16]. This work has been developed at a low level, thus lacking a high abstraction level modeling step, which is the case of most state of the art works in the field. Consequently, to apply our patterns, we start by providing a table of correspondence between adaptation concepts offered by the patterns and those considered in the application example in order to evaluate the generic aspect of the solutions. Then, we present a pattern-based design to show how to append the patterns to the self-adaptive system structure. Since this paper focuses only on the monitoring and analyzing processes, this section is thus limited to the adaptation features related to these two modules.

A. Application description

The application presented in [16] consists in a self-adaptive object tracking application implemented on an FPGA-based smart-camera. The application is composed of 10 tasks which can be implemented in HW or in SW. An electric toy train tracking scenario is proposed to illustrate the system self-adaptivity. The scenario contains various events provoking configuration decisions. The goal is to design an embedded system able to respect a constraint while optimizing secondary magnitudes. In this case study, the regulated magnitude is the QoS indicating the tracking accuracy and the optimized ones are power and execution time.

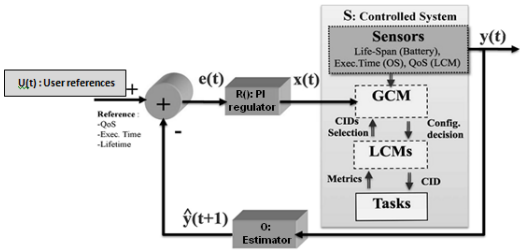

Figure 6. Global structure of the closed-loop self-adaptive system proposed in [16]

This self-adaptive system is developed based on a cross-layer adaptation approach for self-adaptive RTE systems development. Authors propose a hierarchy of local and global configuration managers (LCM/GCM), as depicted in Figure 6, to separately deal with application-specific and application-independent reconfigurations, respectively. The LCM is responsible for local application algorithmic reconfigurations. An LCM is defined for each application. The GCM is responsible for architectural reconfigurations which consist in tasks migration from software to hardware on a multiprocessor heterogeneous architecture. Only one GCM is defined for the whole system.

B. Adaptation concepts correspondence

The adaptation features correspondence is summarized in Table I.

The train tracking application considers three context elements, the QoS metric indicating the tracking error, the execution time and the battery life. It uses three corresponding sensors and a fourth one providing estimations of the context behavior. A LCM compares user-defined references with observed magnitudes values to detect irregular status. It then asks for correction by sending a preliminary decision of application's algorithmic configuration to the GCM. This latter is the responsible for the global reconfiguration decision, taking into account the received LCM local decision. A configuration period Ne is defined to control the periodic execution of the adaptation mechanism. This scenario is very similar, in both structure and behavior, to the RTE Monitor pattern in conjunction with the RTE Analyzer. The LCM encompasses both monitoring and analyzing modules. The three magnitudes are the observed context elements whose measured values, with a set of associated thresholds, form the input of the LCM. Its output is an adaptation request sent to the Decision Maker (the GCM) where the request data is an Algorithmic configuration.

This equivalence study permits to show the important degree of similarity between the proposed patterns and the application case study. That proves the efficiency of the solution in providing a generalized and high abstracted design permitting to cover most RTE adaptation features design.

Table I
ADAPTATION CONCEPTS CORRESPONDENCE BETWEEN RTE MONITOR AND RTE ANALYZER PATTERNS AND THE OBJECT TRACKING APPLICATION

Concept	Instance
Context elements	3 magnitudes: -Application QoS (the tracking error) -Execution time -Power consumption
Sensors	4 observers: -Task T10 -SW timer of RTOS -Battery gauge component -Observer estimator
Monitor + Analyzer	Local configuration manager (LCM)
Monitoring Thresholds - irregular status	QoS reference (the tracking maximum error) is set to 10% and reduced to 2% within the critical area to guaranty good reactivity. Task T10 provides the LCM with the application QoS metric (error between prediction and object position): - A value close to 0 but lower than the reference (10%) means a very high tracking quality –> can be relaxed by reducing the application speed. A value higher than the reference –> the application rate must be increased with a faster configuration.
Decision maker	Global configuration manager (GCM)
Adaptation request (request Data)	An Algorithmic configuration as a first local decision sent by the LCM to the GCM mailbox
System stability and avoidance of reconfiguration	Proportional Integrator (PI) regulator (coefficients: $kp = 0,25; ki = 0,25$), and least mean square (LMS) observer (coefficient $kL = 2^{21}$)
Adaptation period	Configuration period Ne is set to 1, which means (that a new configuration is evaluated after each application iteration

C. Patterns application

The general structure of the closed-loop self-adaptive system is illustrated in Figure 6. It is composed of 4 basic elements:

- The controlled system S is composed of configuration managers, a LCM for the tracking application and a GCM, a set of tasks and sensors observing the controlled magnitude y(t).
- The control function R is a proportional and integrator regulator (PI) permitting to handle system stability and avoidance of reconfiguration.
- The system observer O calculates estimates of controlled magnitude for the next time slot in order to predict its evolution to anticipate the right reconfiguration decision. This model-based estimator permits rapid, accurate and costless estimation of system behavior that replaces new sensors' measures when they are delayed or even not available.
- The user references u(t) represent values or thresholds relative to considered magnitudes, that are defined by the designer to be used by the control mechanism. For instance, a reference can be a lifetime threshold that will be compared to a value provided by the battery gauge component, or an application QoS constraint that will be compared to the QoS value provided by the LCM of the application.

The corresponding high-level pattern-based design is depicted in Figure 7.

D. Discussion

Using the high abstraction level pattern-based modeling for the development of a RTE system presents advantages upon the low-level development approach. Instantiating the patterns for the target system has simplified the design by hiding internal functional details of system elements and lowering the system model size. This follows from the fact that the patterns operate at a high abstraction level to cover adaptation features considered by the low-level system design. In addition, the pattern-based design permitted us to promote modularity and thus flexibility of the design by providing a modular structure for the LCM component.

V. RELATED WORK

The development of self-adaptive systems has begun several years ago and proposed approaches and techniques have evolved with the evolution of technology, system architectures, applications, user requirements and environment constraints. In this section, we limit our review to relevant research works on adaptation of embedded systems, which are the most useful for the definition of our adaptation patterns.

Self-adaptation in embedded systems has been well tack-

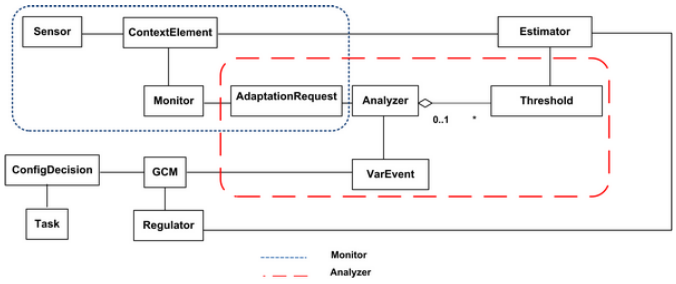

Figure 7. Patterns application

led in the literature. Several approaches have been based on low-level development process integrating adaptation techniques in the classic System on Chip co-design flow [16, 15, 14, 13, 12]. Later, the development of adaptive systems at a low-level has become a tedious task due to the growing complexity of modern systems and dedicated applications. Designers have then resorted to high abstraction level approaches [21, 22, 23, 24], typically based on the model driven engineering (MDE) methodology [25] with the UML/MARTE profile which is the most upcoming standard for embedded systems model-driven development. This methodology has been proven to be well appropriate to embedded system design [26]. It eases the modeling of self-adaptive systems by avoiding dealing with technical details thus promoting reusability. Details about the previously cited works may be found in [4].

In addition to the above approaches, several projects [17, 18] have been realized in the literature to help guide and ease self-adaptive systems development. Researchers have equally developed middleware [19, 27, 28], frameworks [20, 29], languages [30] and tools [31] for this aim. CARISMA (Context-Aware Reflective mIddleware System for Mobile Applications) [27] and MADAM (Mobility and Adaptation Enabling Middleware) [19] offer adaptation middleware to facilitate the development of mobile applications. CARISMA exploits the principal of reflection to enable mobile application to dynamically adapt to context changes. It also offers a conflict resolution approach that treats conflicts that may be incurred by the reflective behavior. The MADAM project proposes a component-based design of both adaptation middleware and mobile applications. Each component has a set of implementations offering the same functional properties but different non-functional ones. The adaptation middleware decides of the implementation that best meets user needs. It is composed of three core components: a context manager which monitors the change in user requirements and context elements, an adaptation manager which takes the appropriate adaptation decisions and a configurator which reconfigures the mobile application according to the adaptation decisions. The author of [20] proposed a model based framework that helps automate the development of self-adaptive embedded systems. He uses formal methods to specify system features which are the embedded system, the events that trigger reconfiguration and the reconfiguration requirements. Author proposes a process formalization that permits to extend the original system model to a self-adaptive one based on its formal specification. The resulting model can be the input of existing model-based simulator or code generator.

All the previously described approaches are beneficial since they facilitate and fasten the development of adaptive systems. However, they present some weaknesses. They are generally domain specific which limits their applicability for diverse systems. They are also not sufficiently generic since they tackle specific adaptation problems, which consequently compromises their reusability as well as their ability to adapt to new system requirements and constraints. Additionally, most of them focus on the software side adaptation while ignoring the hardware side which is essential in the embedded systems design.

The development of design patterns is a promising alternative approach to deal with the above problems. A design pattern gives a higher abstraction view of a commonly recurring problem, thus promoting the reusability and extensibility of the design. Works dealing with pattern-based adaptation are not numerous. Some were interested in defining the internal functioning of adaptation modules [5, 32, 33] while others rather focused on the structure and organization of adaptation functions. Concerning patterns dedicated to the architecture of adaptive systems, we cite Weyns et al. [34] who proposed patterns to decentralize multiple adaptation loops in large and complex self-adaptive systems. In [35] authors proposed a dynamic self-adaptation pattern for distributed transaction management in service-oriented applications (SOA). SOA coordination patterns are used to deal with the coordination of distributed transactions. In [36] a taxonomy was proposed for self-adaptation patterns at both component and ensemble levels. At the component level, authors describe the basic components that may compose self-adaptation patterns. At the ensemble level, mechanisms by which components can be composed into ensembles are presented.

As for patterns dealing with the internals of the adaptation functions, Gamaa and colleagues [5] proposed design patterns to specify the behavior to dynamically reconfiguring four types of software architectures; master/slave, centralized, server/client, and decentralized architectures. Schmidt et al. [32] proposed a set of patterns that can be used for the development of adaptive middleware. For instance, the virtual component pattern [37] permits to adapt a distributed application to embedded systems' memory constraint. The component configurator pattern enables an application to change its components' implementations at run-time. In [33] authors proposed a set of patterns aiming at adapting distributed networked systems in order to satisfy requirements and constraints that arise at execution-time. Patterns are classified, according to their purpose, into three principle categories: monitoring, decision-making and reconfiguration activities.

For example, Sensor Factory pattern is a monitoring pattern dedicated to component-based distributed infrastructure and intends to automatically deploy sensors across a network and probe the distributed components. These patterns are useful for the development of adaptive systems in different domains. However, they do not fit the real-time and embedded systems domain since they do not deal with RTE systems constraints. Also, they are limited to only addressing the software part of the system and are most appropriate for distributed systems.

VI. Conclusion and future work

In this paper, we proposed a generic design of self-adaptive RTE systems based on an adaptation loop composed of four adaptation processes, monitoring, analyzing, deciding and acting, accompanied with sensors and effectors. We detailed the internals of the two first adaptation processes. We proposed design patterns giving generic solutions for the monitoring of an RTE system context and the verification of its constraints meeting to decide about the need of an adaptation action. Both patterns deal with the system stability issue by minimizing events trigger through the selection of only important context variations. In addition, they handle concurrency and real-time features of adaptation operations. The proposed solution permits to guide adaptive systems designers decrease the complexity of their heavy job and fasten the development of such systems by promoting reusability of the design.

The proposed patterns were applied to a RTE system running a self-adaptive object tracking application implemented on an FPGA-based smart-camera. The case study shows the effectiveness of using high abstraction level pattern-based modeling for the development of a RTE system. It promotes modularity and flexibility of the design by providing a modular structure.

We plan in future work to complete the modules of the adaptation loop. Indeed, our goal is to integrate the proposed patterns in an MDE-based approach for the automatic generation of self-adaptive RTE systems.

References

[1] Svein Hallsteinsen, Erlend Stav, and Jacqueline Floch. Self-adaptation for everyday systems. In *Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*, WOSS '04, pages 69–74, New York, NY, USA, 2004. ACM.

[2] Anders Kofod-Petersen and Marius Mikalsen. Context: Representation and reasoning. representing and reasoning about context in a mobile environment. *Revue d'Intelligence Artificielle*, 19(3):479–498, 2005.

[3] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2):14:1–14:42, May 2009.

[4] Mouna Ben Said, Yessine Hadj Kacem, Nader Ben Amor, and Mohamed Abid. High level design of adaptive real-time embedded systems: A survey. In *International Conference on Model-Driven Engineering and Software Development - MODELSWARD 2013*, pages 341 – 350, 19-21, February, 2013.

[5] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[6] OMG Object Management Group. A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, ptc/2011-06-02. Object Management Group, June June 2011.

[7] Peyman Oreizy, Michael M. Gorlick, Richard N. Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S. Rosenblum, and Alexander L. Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3):54–62, May 1999.

[8] Jesper Andersson, Rogério Lemos, Sam Malek, and Danny Weyns. Software engineering for self-adaptive systems. chapter Modeling Dimensions of Self-Adaptive Software Systems, pages 27–47. Springer-Verlag, Berlin, Heidelberg, 2009.

[9] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[10] Philip K. McKinley, Seyed Masoud Sadjadi, Eric P. Kasten, and Betty H. C. Cheng. Composing adaptive software. *Computer*, 37(7):56–64, July 2004.

[11] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-oriented software architecture: a system of patterns*. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[12] Nam Pham Ngoc, Wolfgang van Raemdonck, Gauthier Lafruit, Geert Deconinck, and Rudy Lauwereins. A qos framework for interactive 3d applications. In *WSCG*, pages 317–324, 2002.

[13] Wanghong Yuan and Klara Nahrstedt. Energy-efficient cpu scheduling for multimedia applications. *ACM Trans. Comput. Syst.*, 24(3):292–331, August 2006.

[14] Vibhore Vardhan, Wanghong Yuan, Albert F. Harris III, Sarita V. Adve, Robin Kravets, Klara Nahrstedt, Daniel Grobe Sachs, and Douglas L. Jones. Grace-2: integrating fine-grained application adaptation with global adaptation for saving energy. *IJES*, pages 152–169, 2009.

[15] Linfeng Ye, Jean-Philippe Diguet, and Guy Gogniat. Rapid application development on multi-processor reconfigurable systems. In *FPL*, pages 285–290, 2010.

[16] Jean-Philippe Diguet, Yvan Eustache, and Guy Gogniat. Closed-loop–based self-adaptive hardware/software-embedded systems: Design methodology and smart cam case study. *ACM Trans. Embed. Comput. Syst.*, 10(3):38:1–38:28, May 2011.

[17] Famous Project. Anr famous overview.

http://www.lifl.fr/ meftali/famous/.

[18] Nishanth Shankaran, John S. Kinnebrew, Xenofon D. Koutsoukas, Chenyang Lu, Douglas C. Schmidt, and Gautam Biswas. An integrated planning and adaptive resource management architecture for distributed real-time embedded systems. *IEEE Trans. Comput.*, 58(11):1485–1499, November 2009.

[19] Marius Mikalsen, Nearchos Paspallis, Jacqueline Floch, Erlend Stav, George A. Papadopoulos, and Akis Chimaris. Distributed context management in a mobility and adaptation enabling middleware (madam). In *SAC*, pages 733–734, 2006.

[20] Li Tan. Model-based self-adaptive embedded programs with temporal logic specifications. In *Proceedings of the Sixth International Conference on Quality Software*, QSIC06, pages 151–158, Washington, DC, USA, 2006. IEEE Computer Society.

[21] Imran Rafiq Quadri, Huafeng Yu, Abdoulaye Gamatié, Eric Rutten, Samy Meftali, and Jean-Luc Dekeyser. Targeting Reconfigurable FPGA based SoCs using the MARTE UML profile: from high abstraction levels to code generation. *Special Issue on Reconfigurable and Multicore Embedded Systems, International Journal of Embedded Systems (IJES)*, 2010.

[22] Jorgiano Vidal, Florent de Lamotte, Guy Gogniat, Jean-Philippe Diguet, and Philippe Soulard. Uml design for dynamically reconfigurable multiprocessor embedded systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE 10, pages 1195–1200, 3001 Leuven, Belgium, Belgium, 2010. European Design and Automation Association.

[23] Fatma Krichen, Brahim Hamid, Bechir Zalila, and Mohamed Jmaiel. Towards a model-based approach for reconfigurable dre systems. In *ECSA*, pages 295–302, 2011.

[24] Ji Zhang and Betty H. C. Cheng. Model-based development of dynamically adaptive software. In *Proceedings of the 28th international conference on Software engineering*, ICSE '06, pages 371–380, New York, NY, USA, 2006. ACM.

[25] Douglas C. Schmidt. Model-driven engineering. *IEEE Computer*, 39(2), February 2006.

[26] Guy Gogniat, Jorgiano Vidal, Linfeng Ye, Jérémie Crenne, Sébastien Guillet, Florent de Lamotte, Jean-Philippe Diguet, and Pierre Bomel. Self-reconfigurable embedded systems: From modeling to implementation. In *ERSA*, pages 84–96, 2010.

[27] L. Capra, W. Emmerich, and C. Mascolo. Carisma: context-aware reflective middleware system for mobile applications. *Software Engineering, IEEE Transactions on*, 29(10):929–945, 2003.

[28] S. M. Sadjadi and P. K. McKinley. A survey of adaptive middleware. Technical Report MSU-CSE-03-35, Department of Computer Science, Michigan State University, East Lansing, Michigan, December 2003.

[29] Wanghong Yuana, Klara Nahrstedta, Saritav. Advea, and Douglas L. Jonesb. Design and evaluation of a cross-layer adaptation framework for mobile multimedia systems. In *Proc. SPIE 5019, Multimedia Computing and Networking*, 2003.

[30] Thomas Vogel and Holger Giese. Model-driven engineering of adaptation engines for self-adaptive software: Executable runtime megamodels. Technical Report 66, Hasso Plattner Institute for Software Systems Engineering, University of Potsdam, Germany, 4 2013.

[31] Nelly Bencomo, Paul Grace, Carlos Flores, Danny Hughes, and Gordon Blair. Genie: supporting the model driven development of reflective, component-based adaptive systems. In *Proceedings of the 30th international conference on Software engineering*, ICSE '08, pages 811–814, New York, NY, USA, 2008. ACM.

[32] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. Pattern-oriented software architecture. In *Volume 2: Patterns for Concurrent and Networked Objects*. Wiley, 2000.

[33] Andres J. Ramirez and Betty H. C. Cheng. Design patterns for developing dynamically adaptive systems. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '10, pages 49–58, New York, NY, USA, 2010. ACM.

[34] Danny Weyns, Bradley Schmerl, Vincenzo Grassi, Sam Malek, Raffaela Mirandola, Christian Prehofer, Jochen Wuttke, Jesper Andersson, Holger Giese, and KarlM. Göschka. On patterns for decentralized control in self-adaptive systems. In Rogério Lemos, Holger Giese, HausiA. Müller, and Mary Shaw, editors, *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *Lecture Notes in Computer Science*, pages 76–107. Springer Berlin Heidelberg, 2013.

[35] H. Gomaa and K. Hashimoto. Dynamic self-adaptation for distributed service-oriented transactions. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on*, pages 11–20, 2012.

[36] Mariachiara Puviani, Giacomo Cabri, and Franco Zambonelli. A taxonomy of architectural patterns for self-adaptive systems. In *Proceedings of the International C* Conference on Computer Science and Software Engineering*, C3S2E '13, pages 77–85, New York, NY, USA, 2013. ACM.

[37] Angelo Corsaro, Douglas C. Schmidt, Raymond Klefstad, and Carlos ORyan. Virtual component - a design pattern for memory-constrained embedded applications. In *in Proceedings of the Ninth Conference on Pattern Language of Programs (PLoP*, 2002.