A UML-Profile for domain specific patterns: Application to real-time

Saoussen Rekhis[1], Nadia Bouassida[2], Rafik Bouaziz[1], Bruno Sadeg[3]

[1,2] MIRACL-ISIMS, Sfax University, BP 1088, 3018, Sfax, Tunisia.
[3]LITIS, UFR des Sciences et Techniques, BP 540, 76 058, Le Havre Cedex, France.

[1]{saoussen.rekhis, raf.bouaziz}@fsegs.rnu.tn
[2]nadia.bouassida@isimsf.rnu.tn
[3]bruno.sadeg@univ-lehavre.fr

Abstract. The design of Real-Time (RT) applications is a difficult task since it must take into account the specification of time-constrained data and time-constrained transactions. The design of these applications can be facilitated through the reuse of RT design patterns that improve software quality and capture RT domain knowledge and design expertise. However, the difficulty of RT design patterns comprehension reinforces the need for a suitable design language. This language has to express concepts modeling RT features and distinguishing the commonalities and differences between RT applications.

This paper presents new UML notations that take into account the design of both RT specific concepts and the variability of domain specific patterns. The UML extensions are, then, illustrated in the RT context using an example of a controller pattern.

Keywords: UML notation, domain specific patterns, instantiation, real-time applications.

1 Introduction

A design pattern [1] is a description of a solution to a common problem in software design. It captures the design expertise necessary for developing applications and allows the reuse at both the design and code levels. Design patterns can be general and cover different domains of application (e.g. patterns of GoF [1]) and they can, also, be intended for a particular domain, in this case they are called domain-specific patterns [24].

Despite their advantages, to benefit from design patterns, a designer must spend a lot of time in understanding and then reusing the design pattern in a certain application. To facilitate the reuse and instantiation phase, many design pattern notations have been proposed ([8], [4], [3]). The proposed approaches offer essentially UML extension mechanisms such as stereotypes, tags and constraints to cope with the pattern variability and to show the pattern specificities.

I. Reinhartz-Berger, A. Sturm, Y. Wand, J. Bettin, T. Clark, S. Cohen, J. Ralyté, and P. Plebani (Eds.):
CAiSE 2010 Workshop DE@CAiSE'10, Hammamet, Tunisia, pp. 32-46, 2010.

These design languages with their UML extensions remain insufficient when they deal with a specific domain. In fact, in the design of a specific domain, the design language has to take into account not only the variability and the aspects relative to the pattern, but also the extensions and specificities of the domain itself. For example, when considering the Real Time (RT) domain, we found that this domain has many details that must be taken into account by the design pattern notation.

In fact, RT applications, which manipulate voluminous quantities of data, have two main features: i) they manipulate RT data that must closely reflect the current state of the controlled environment, and ii) they must be able to meet RT constraints of transactions. These two features must be considered by RT design patterns.

This paper proposes a new UML-profile that extends UML with concepts related to RT design patterns. The motivations behind these extensions are three-folds. The first motivation is to have flexible patterns that distinguish the fixed parts from the optional and variable elements in the pattern. The second motivation is to facilitate the comprehension of design patterns instantiation and to guide a designer to derive a specific application. The third motivation is to present design patterns for the RT domain using the proposed profile which is extended with RT specific concepts.

The remainder of this paper is organized as follows. Section 2 overviews and evaluates currently proposed design languages and their extensions. Section 3 presents our proposition to represent an UML profile for RT design patterns. Section 4 illustrates the design language with a RT controller pattern and presents an example of a freeway traffic management system reusing it. Section 5 concludes the paper and outlines future work.

2 Overview of current works

In order, to propose a RT pattern profile, we have been inspired in our work from RT profiles and existing pattern notations. Thus, in this section we, first, overview current design languages for pattern's representation. For this reason, we define a set of criteria necessary for pattern notations and then we present their advantages and limits. Second, we briefly present in Subsection 2.2 the RT profiles and the UML extensions taking into account the real-time system requirements.

2.1 Overview of UML extensions for design patterns representation

Several criteria have to be taken into account to evaluate the currently proposed languages for pattern representations. These criteria are used to compare current UML-based pattern notations, for the specification of general and domain-specific design patterns and for their instantiation.

- Criteria for design pattern representation at the specification level

C1. Expressivity: Design patterns have mostly been described using natural language, complex mathematical or logic based formalisms [5] [6] which are not easily

Table 1. Comparison of current notations using the specification criteria.

	Design pattern specification criteria		
	Expressivity	**Variability**	**Definition of constraint**
Dong & Yang UML profile [3]	This profile proposes notations that focus more on the pattern applicability context than on the pattern specification.	Unlike several others notations [8] [4], the proposed profile does not focus on specifying the variability of a pattern solution.	These notations don't specify constraints which delimit the pattern applicability.
P_UML profile [8]	P_UML proposes extensions showing the pattern hot-spots in a class diagram and guiding the designer in instantiating a pattern. However, it does not distinguish between the extensions used in pattern instantiation from those used in pattern specification, which reduces the expressivity of notations.	This profile is characterized by: -The definition of tagged values to extend the static view: {*variable*} indicates that the method implementation varies according to the pattern instantiation; {*extensible*} indicates that the class interface may be extended by adding new attributes and/or methods; -The applicability of the {*incomplete*} constraint on generalization relation to indicate that new classes may be added during the pattern instantiation	These notations propose to define the pattern constraints through notes containing OCL constraints.
Arnaud profile [4]	This profile is not very expressive since the static view of a pattern is presented by very elementary separated packages which contain one or two classes. This reduces the understanding and makes the composition more difficult.	Unlike all previous notations, this profile focuses on the variability in the functional, dynamic and static views. The use case diagram is the entrance point for the instantiation process, where the application designer selects a functionality variant. However, the use case diagram is too abstract and can not be used as an input model for the patterns instantiation. In fact, the use case diagram is at a high level of abstraction and thus the designer cannot identify, for example, the optional attributes or methods according to its needs.	Similar to P_UML, this profile uses notes that contain OCL constraints. These latter must be fulfilled by a pattern to be applied correctly.
ADOM-UML [18]	ADOM-UML is an Application based DOmain Modeling approach, in which UML 2.0 is used as the modeling language of both: the domain and	ADOM-UML defines new stereotypes in order to denote the multiplicity variability of the different domain model elements. The multiplicity stereotypes aim to represent how many times a model element can appear in a	The constraints are well defined in ADOM-UML among the different layers: the domain layer enforces

	element if their meta-classes in the language layer are the same (e.g., a class that appears in a domain model may serve as a classifier of classes in an application model). Thereby, ADOM-UML provides support for traceability criterion and enhances the readability of an application model.	according to the adaptation of one domain model and doesn't deal with the composition of many reusable domain artefacts, such as patterns.

In summary, none of the proposed notations satisfies all the different specification and instantiation criteria, when representing patterns. Moreover, none of them proposes extensions showing the behavioral composition.

2.2 Overview of UML extensions for RT applications

Several works have proposed UML extensions to take into account the real-time system requirements such as, *RT-UML* [20] and *ACCORD/UML* [21]. The basic concepts of *RT-UML* were integrated in the UML standard through the UML profile for **S**chedulability, **P**erformance, and **T**ime (denoted SPT profile) [22]. Recently, *MARTE* profile [10] for **M**odeling and **A**nalysis of **R**eal-**T**ime **E**mbedded systems has been standardized by the OMG. It is intended to replace the existing UML Profile for SPT profile [22]. MARTE consists in defining extensions that provide high-level modelling concepts to deal with RT and embedded features modeling as well as specific modeling artifacts to be able to describe both software and hardware execution supports.

Another work proposed the *UML-RTDB* profile [23] to express real-time database features in a structural model. Unlike the previous profiles, it supplies concepts for real-time database modeling such as RT attributes, RT methods and RT classes. In addition, UML-RTDB specifies two kinds of real-time attributes, *sensor* attributes and *derived* attributes, in order to satisfy the requirements of current real-time applications. However, some proposed stereotypes overlap with the UML extensions presented by MARTE profile especially those relative to the RT methods. In fact, the UML-RTDB stereotypes *<<Periodic>>*, *<<Sporadic>>* and *<<Aperiodic>>* that express respectively periodic, sporadic and aperiodic methods in the class diagrams, has the same meaning as the tagged value *Occurrence Kind* of the *<<rtFeature>>* stereotype defined in MARTE. Thereby, we adapt some MARTE stereotypes modeling RT aspects instead of the other UML extensions proposed for the modeling of RT applications since MARTE is a standardized profile.

Nevertheless, the only use of UML notations modeling RT application characteristics is insufficient to specify RT design patterns. That is, RT patterns must be generic designs intended to be specialized and reused by any application in RT domain. For this reason, in addition to the UML extensions representing RT aspects, we need new notations distinguishing the commonalities and differences between applications in the pattern domain. Moreover, we need new concepts for the explicit representation of the pattern elements roles for the traceability purpose.

In the next section, we describe the extensions that we propose to take into account these new concepts.

• **Stereotype** <<***variable***>> (applied to the *Operation* UML Metaclass): This stereotype has the same meaning with the {variable} tagged value proposed in [8]. It indicates that the method implementation varies according to the pattern instantiation.

3.2 UML Extensions for instantiating domain-specific patterns

Some of the existing notations (Dong & Yang UML profile [3] and P-UML profile [8]) provide support on how to keep trace of the pattern when instantiated. These notations focus only on generic design patterns for which it is difficult to recognize the pattern instance when it is composed with others in a particular design. Thus, it is essential to hold the pattern name and the role played by each element (class, attribute and method) in the instantiation.

However, a domain specific pattern is instantiated in the scope of a domain. Therefore, it is easy to retrieve the pattern-related information even after the pattern is applied or composed with other patterns. We assume that omitting both the name and the role of pattern attributes and operations will not create any ambiguity. For this reason, we propose to present only the pattern name and the role names of the classes in order to avoid overloaded models. In fact, pattern-related information should be minimized in the class and sequence diagrams for readability [3].

We propose to define two new stereotypes for the explicit visualization of patterns in an application design:

• <<***patternClass***>> stereotype: It is applied to the *Class* UML metaclass in order to indicate that it is an instantiated pattern class and not originally defined by the designer. We propose to define two properties related to this stereotype:
- *patternName* tag : indicates the pattern name,
- *participantRole* tag : indicates the role played by the class in a pattern instance.

• <<***patternLifeline***>> stereotype: It is applied to the *Lifeline* metaclass in order to distinguish between the objects instantiated from the pattern sequence diagram and those defined by the designer. This stereotype has the same properties than <<patternClass>> stereotype.

These stereotypes allow to eliminate any confusion when patterns are composed. That is, when two or more classes represent the overlapping part of the composition, the proposed stereotype shows the roles that these classes play in each pattern.

3.3 UML extensions for modeling RT aspects

In addition to the above described stereotypes distinguishing the fixed parts from the optional and variable parts in the pattern, the specification of RT design patterns needs UML extensions supporting the modeling of RT aspects. Thus, we import stereotypes from HLAM (High Level Application Modeling) and NFP (Non Functional Properties) sub-profiles of MARTE [10] (cf. figure 1). Note that MARTE provides support required from specification to detailed design of RT embedded systems characteristics. However, only the extensions describing RT applications features at a high level of

4 A RT design pattern example

In this section, we propose to illustrate the proposed extensions through an example of a reusable RT design pattern that explicitly shows the generic data which are fundamental and which represent the core of RT applications, on the one hand, and the allowed variants, on the other hand.

4.1 RT controller pattern

RT applications perform several RT processes among which: the RT data acquisition and the data control processes. We focus in this paper on modeling the static as well as the dynamic view of RT data control process through the definition of RT controller pattern.

- Interface:

Name: controller pattern

Context: This pattern is applicable in all RT applications which need to be managed by Real Time Database (RTDB) systems. In fact, a RTDB has all the requirements of traditional databases, but it also requires management of time-constrained data and time-constrained transactions [11].

Intention: The pattern aims to model the control of the data acquired from environment and the initialization of corrective action(s) if a violation is found.

- Solution:

Static specification: Figure 2 presents the controller pattern static view.

Participants*:*

- Observed_element: This class represents the description of a physical element that is supervised by the controller. It can be an aircraft, a car, a road segment, and so on. One or more measure types (i.e. Temperature, Pressure, etc) of each observed element could determinate its evolution. These measures are classified into either base measures or derived measures. Base measures stand for RT data that are issued from sensors, whereas derived measures stand for RT data that are calculated by the controller using base measures. The refreshment of each derived RT data is required every time one of the base data is updated.

The *ObservedElement* class has the *ElmentID* and *ElementStatus* fundamental attributes. In addition, it has an *UpdateStatus ()* method allowing to update the status of observed element according to the variation of the captured values.

- Controller: A controller has to monitor physical elements for responding to conditions that might violate safety. It takes periodically the value captured for each observed element as well as the minimum value and the maximum value that define the interval for which the controller does not detect an anomaly. If a captured value does not verify the boundary constraint, then the controller initiates some corrective actions, such as a reset and a shut-down, or sends an alarm to notify an operator.

On the other hand, the controller receives periodically an update message from an observed element to notify it about the modification of its measures. In this case, the

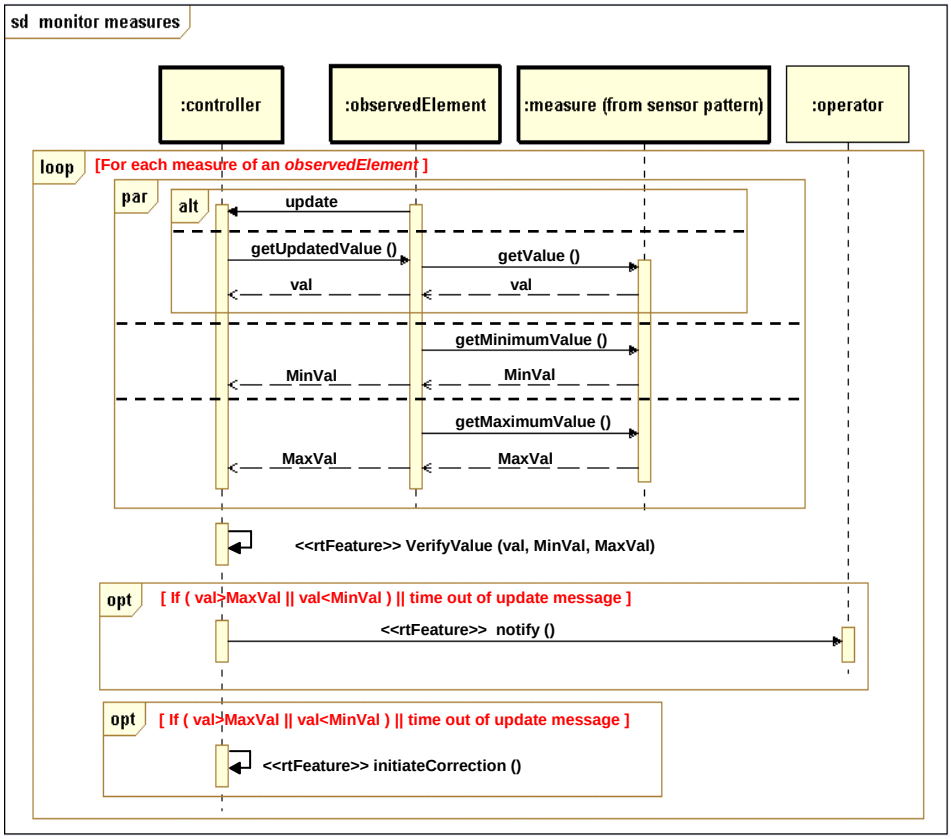

Fig 3. Specification of RT controller pattern dynamic view

4.2 RT sensor pattern instantiation: an example

This section proposes to illustrate the reuse of RT controller pattern through the design of freeway traffic management system.

The increasing road transport traffic and the incessant rise of the number of vehicles have caused a great growth of the magnitude of traffic flows on public roads. In consequence, freeway traffic management systems have become an important task intended to improve safety and provide a better level of service to motorists. We describe, in the following an example of a freeway traffic management system: COMPASS [19]. We focus precisely on modeling the compass control data subsystem and we explain how this design issue can be facilitated by the reuse of the RT controller pattern.

The current traffic state is obtained from the essential sources: inductance loop detectors and supervision cameras. In fact, vehicle detector stations use inductance loops to measure speeds and lengths of vehicles, traffic density (i.e. number of vehicles in a road segment) and occupancy information. Whereas, the supervision cameras are

In order to represent RT design patterns in a more readable manner, this paper proposed UML-based extensions distinguishing clearly between the different parts constituting the pattern. These extensions help the designer in determining the variable elements that may differ from one application to another and allows to identify, easily, design patterns when they are applied to model a particular RT application. Besides, this paper proposed to guide the designer in modeling features specific to RT domain through the use of stereotypes imported from MARTE profile. These stereotypes provide facilities to model RT applications characteristics at a high abstraction level, being independent from the nature of tools used for the implementation of RT systems. The paper illustrated the proposed notations through the specification of RT controller pattern and its instantiation to design a freeway traffic management system.

Our future works include two axes. Firstly, we are looking into the formalization of RT design patterns. Secondly, we must examine how to integrate the design patterns in the context of the model driven architecture in order to add more assistance when generating models by reusing patterns. This could bring new benefits and impulse for both the knowledge capturing techniques and the software development process quality.

References

1. Gamma E., Helm R., Johnson R.E, Vlissides J., Design patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Edition, 1994.
2. Fowler M., Analysis Patterns – Reusable Object Models, Addison-Wesley, 1997.
3. Dong J. and Yang S., Visualizing design patterns with a UML profile, proceedings of IEEE Symposium on Human Centric Computing Languages and Environments, pp: 123-125, 2003.
4. Arnaud N., Front A.and Rieu D., Expression et usage de la variabilité dans les patrons de conception, Revue des sciences et technologies de l'information, série : Ingénierie des Systèmes d'Information, vol. 12/4, pp. 21-24, 2007.
5. Eden A.H., Gil J., Hirshfeld Y., Yehudai A., Towards a mathematical foundation for design patterns, Technical report, dept.of information technology, U.Uppsala, 1999.
6. Mikkonen T., Formalizing Design Patterns, Proc. 20th International Conference on Software Engineering— ICSE, pp. 115–124, 1998.
7. OMG, UML 2.0 OCL specification, 2003.
8. Bouassida N., Ben-Abdallah H., Extending UML to guide design pattern reuse, Sixth Arab International Conference On Computer Science Applications, Dubai, 2006.
9. OMG, Unified Modeling Language (UML) Infrastructure: v2.1.2, formal/2007-11-04, 2007.
10. OMG, A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, OMG document number: ptc/2008-06-09, 2008.
11. Ramamritham K., Real-Time Databases. Journal of Distributed and Parallel Databases, 1(2):199–226, 1993.
12. Ramamritham K., Son S., and DiPippo L., Real-Time Databases and Data Services. Real-Time Systems, 28:179–215, 2004.
13. Kim D.K., France R., and Ghosh S., A UML-based language for specifying domain-specific patterns, Journal of Visual Languages and Computing, pp. 265–289, 2004.
14. Douglass B. P., Real-Time Design Patterns: Robust Scalable Architecture for Real Time Systems, Addison-Wesley Edition, September 27, 2002
15. M. Amirijoo, J. Hansson, and S. H. Son. Specification and management of QoS in real-time databases supporting imprecise computations. IEEE Transactions on Computers, 55(3), 2006.