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 understood by an inexperienced designer. This leads to complications in incorporating design patterns effectively into the modelling of a new system. To remediate to this difficulty, the solution is using an expressive visual notation based on UML to specify patterns. This improves the pattern specification quality because UML allows to easily visualise, define and document the artefacts of the system under development.

C2. Variability: The design patterns have to incorporate flexibility and variability in order to guide the designer in determining the variable elements that may differ from one application to another. In fact, variability is classified into optional and alternative characteristics. So, it is important to show the optional elements which can be omitted in a pattern instance. It is also necessary to clarify the variability points (called *hot-spots*) which describe the elements that can vary according to a specific context.

C3. Constraints definition: The correct instantiation of patterns is a major problem when we want to design a new system by composing design patterns. The validity of an instantiation depends on respecting the properties inherent to the solution. These properties are specified by constraints that are generally expressed in OCL (Object Constraint Language) [7]. They are presented on the class diagram using *notes*.

- Criteria for design pattern representation at the instantiation level

C1. Traceability: The traceability consists of easily identifying design patterns when they are applied and composed with other patterns. In fact, we not only need to identify each pattern in a design, but also we want to show the methods and attributes that play important roles in the pattern. Explicit representation of the key methods and attributes can assist on the traceability of a pattern since it allows us to trace back to the design pattern from a complex design diagram [8].

C2. Composition: The development of applications using design patterns as design components requires a careful look at composition techniques, which are categorized as: behavioural composition techniques and structural composition techniques. Indeed, the behavioural techniques show how dynamic specifications of patterns can be composed using sequence diagram, whereas structural techniques show how the static architectural specifications of instantiated patterns can be composed using class diagram [16].

- UML notations for design patterns

There are several UML notations which proposed extensions to present general design patterns and domain models. Many of them can be used to express concepts relative to domain-specific design patterns such as their flexibility. A comparison of the most recent notations, using the specification and instantiation criteria is proposed in Tables 1 and 2.

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

	application models. Unlike the previous works [3] [4] [8], the ADOM-UML enhances the expressivity of the proposed notations since it well differentiates between the extensions used in the language, domain and application layers. This means that each layer includes modeling constructs that will be used in the more specific layer.	specific context. Particularly, the authors define four stereotypes: <<optional single>>, <<optional many>>, <<mandatory single>> and <<mandatory many>>. Each stereotype has two associated tagged values, min and max, which define the lowest and the upper most multiplicity boundaries. However, the word 'many' used in these stereotypes doesn't enhance the semantic of UML model since each element in a model can be instantiated implicitly many times.	constraints on the application layer, while the language layer enforces constraints on both domain and application layers. Besides, ADOM-UML specifies additional constraints and dependencies in the domain layer expressed in OCL.

Table 2. Comparison of current notations using the instantiation criteria.

	Design pattern instantiation criteria	
	Traceability	**Composition**
Dong & Yang UML profile [3]	This profile proposes new stereotypes and tagged values for the explicit representation of design patterns in software designs. These extensions show the pattern name, the role names of the classes, the attributes and the operations in the pattern and how many instances of a design pattern are applied.	This profile deals with the composition of patterns statically. That is, when two or more classes represent the overlapping part of the composition, the proposed notation shows the roles that these classes play in each pattern.
P_UML profile [8]	This notation proposes to show the pattern participant roles by using an ellipse in the bottom of a class that indicates the pattern name and the role through which this class participates in the pattern. Thereby, it provides support for traceability of pattern instantiation. However, the class diagram may seem to be overloaded since the notation presents an association between ellipses to join the elements of the same pattern.	Like the previous work [3], this profile proposes extensions showing the composition of patterns presented by class diagrams. It does not present notations to deal with the composition of patterns dynamic specifications.
Arnaud & al. UML profile [4]	This profile defines a process to show the steps of patterns instantiation. However, it does not permit the visualization and preservation of pattern-related information in patterns instances in a design model. Consequently, it does not deal with the traceability criterion.	This profile does not present mechanisms to compose neither static, nor dynamic specifications of patterns.
ADOM-UML [18]	The connection between the domain and application layers is done through the stereotypes extension mechanism. This means that a domain element can serve as a stereotype of an application	The composition criterion is not taken into account in domain models. In fact, an application model is created

	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.

3 The UML profile for RT design patterns

In the present work, we extend the unified modeling language “UML 2.1.2” [9] to represent design patterns for RT applications. These extensions allow (i) to express the variability in a pattern, (ii) to identify the roles played by each pattern element in the application instantiating it and (iii) to specify RT applications constraints and their non functional properties. The proposed extensions are described in the next section.

3.1 UML extensions for specifying domain-specific patterns

In this section, we propose new stereotypes showing the optional and fundamental elements participating in a pattern and assisting the designer in pattern reuse. Thus, the class diagram Metamodel is extended with the following stereotypes:

- **Stereotype <<*optional*>>** (applied to the *Feature* UML Metaclass): This stereotype is inspired from <<optional single>> and <<optional many>> stereotypes defined in [18]. In fact, the variety of applications within the RT domain is quite large. For this reason, we can not specify **exactly** how many times a pattern element can appear in a specific RT application. Thus, we use <<optional>> stereotype to represent the optional features (i.e. attribute or method) that can be omitted in a pattern instance.
Each method or attribute which is not stereotyped <<optional>> in a fundamental classifier (i.e. class, interface …) means that it is an essential element that plays an important role in the pattern.

- **Stereotype <<*mandatory*>>** (applied to the UML Metaclasses: *Class*, *Association*, *Interface, Lifeline and ClassAssociation*): This stereotype is inspired from <<mandatory single>> and <<mandatory many>> defined in [18]. We propose the <<mandatory>> stereotype to specify a fundamental element (association, aggregation,…) that must be instantiated at least once by the designer when he models a specific application. For the clarity purpose, a fundamental element in the pattern is drawn with a highlight line like this class ▤.

Besides, each pattern element which is not highlighted means that it is an optional one, except the generalization relation that permits to represent alternative elements. All the attributes and methods of an optional class are implicitly optional.

- **Stereotype <<*extensible*>>** (applied to the UML Metaclasses: *Class*, *Interface and ClassAssociation*): This stereotype is inspired from {extensible} tagged value proposed in [8]. It indicates that the class interface may be extended by adding new attributes and/or methods. Moreover, two properties related to the extensible stereotype are proposed, in order to specify the type of features (attribute or method) that may be added by the designer.
  - *extensibleAttribute* tag: It takes the value *false*, to indicate that the designer cannot add new attributes when he instantiates the pattern. Otherwise, this tag takes the value *true*.
  - *extensibleMethod* tag: It indicates if the designer may add new methods when he instantiates the pattern. The default value is *true*.

- **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 abstraction are taken into account since RT patterns can be instantiated to model many RT applications and not only the embedded systems.

From HLAM sub-profile, we import the <<rtFeature>> stereotype in order to model temporal features. This stereotype extends the metaclasses: message, action, signal and behavioral features. It possesses nine tagged values among which: relD1 (i.e. specification of a relative deadline), absD1 (i.e. specification of an absolute deadline), Miss (i.e. percentage of acceptance for missing the deadline), occKin (i.e. specification of the type of event: periodic, aperiodic or sporadic)… . We propose to annotate each model element that has real-time features with the previously described stereotype.

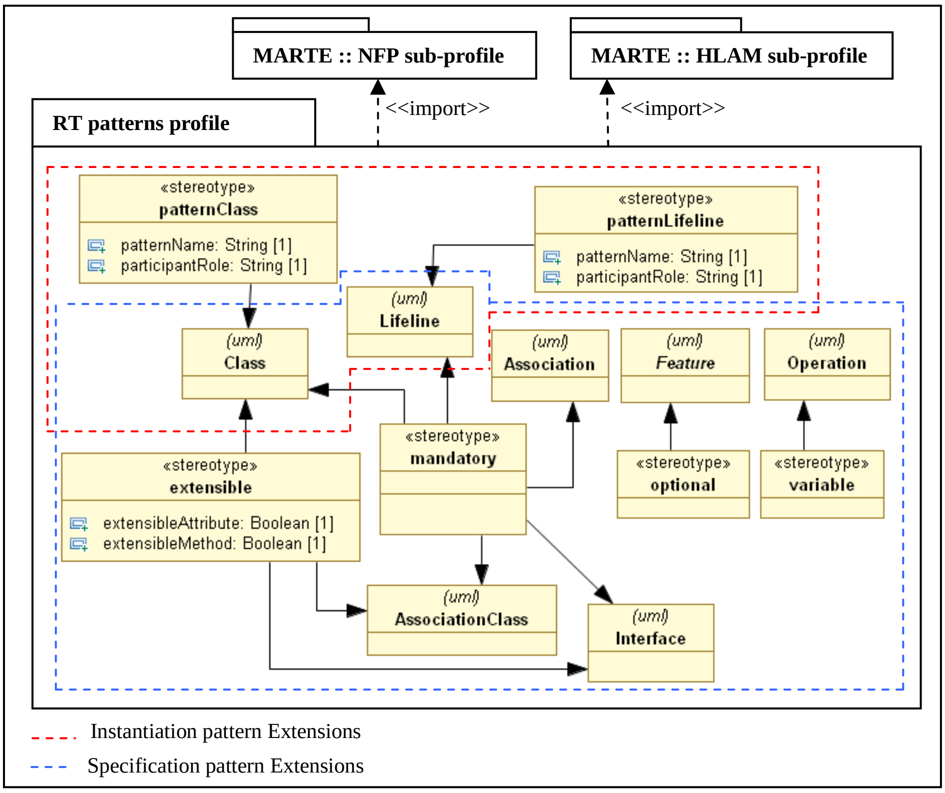

Fig 1. RT pattern profile Metamodel

From NFP Modeling sub-profile of MARTE, we import two stereotypes: <<Nfp>> and <<NfpType>>. The first one extends the Property metaclass. It shows the attributes that are used to satisfy non functional requirements. The second stereotype extends the DataType metaclass. There is a set of pre-declared *NFP_Types* which are useful for specifying NFP values, such as NFP_Duration, NFP_DataSize and NFP_DataTxRate.

In the following section, we illustrate the RT design pattern profile through the specification of RT controller pattern.

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 controller is waiting for a message. If this message does not arrive on time, then the controller performs appropriate recovery actions [14].

As illustrated in Figure 2, the controller class has four methods. The only fundamental method is *VerifyValue()*since it is essential to check that the boundary constraints are fulfilled for all RT applications. This method is performed periodically. In addition, it must be achieved before a deadline. Thus, the *VerifyValue()*method is stereotyped <<rtFeature>> in order to define the periodicity, the relative and absolute deadlines that are tagged respectively period, relDl and absDl. The method *CalculateDerivedValue()* is optional since it can be omitted in a pattern instantiation, when the designed application does not have derived measures. It is stereotyped <<rtFeature>> since it is sporadic and has to meet the deadline defined by the designer. The methods *notify()*, *initiateCorrection()* are optional since the choice of the appropriate recovery action depends on the application instantiating the pattern.

- Operator: The alarm signals sent by the controller are supervised by the operators. These latter provide decisions to validate reported incidents in case the controller only reports errors and does not have the responsibility to take further actions; or in case the confirmation of an operator is needed to achieve the correction.

The Operator class is optional since the controller can take the correction initiative without the intervention of an operator.

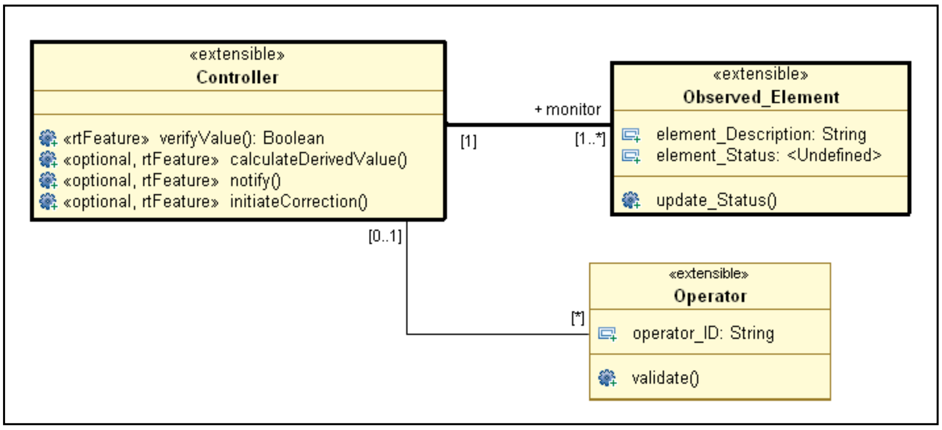
«extensible»
Controller
«rtFeature» verifyValue(): Boolean
«optional, rtFeature» calculateDerivedValue()
«optional, rtFeature» notify()
«optional, rtFeature» initiateCorrection()
+ monitor
[1]
[1..*]
«extensible»
Observed_Element
element_Description: String
element_Status: <Undefined>
update_Status()
[0..1]
[*]
«extensible»
Operator
operator_ID: String
validate()

Fig 2. Specification of RT controller pattern static view

Dynamic specification: Figure 3 presents the controller pattern dynamic view.

In order to verify the validity of each observed element measure, the controller takes the current captured value and the value thresholds in parallel. Then, it verifies that each measured value is in the closed range [Minimum-Value, Maximum-value]. If this constraint is violated or the update message received from an observed element occurs too late, then the controller notifies the operator or initiates the appropriate recovery actions.

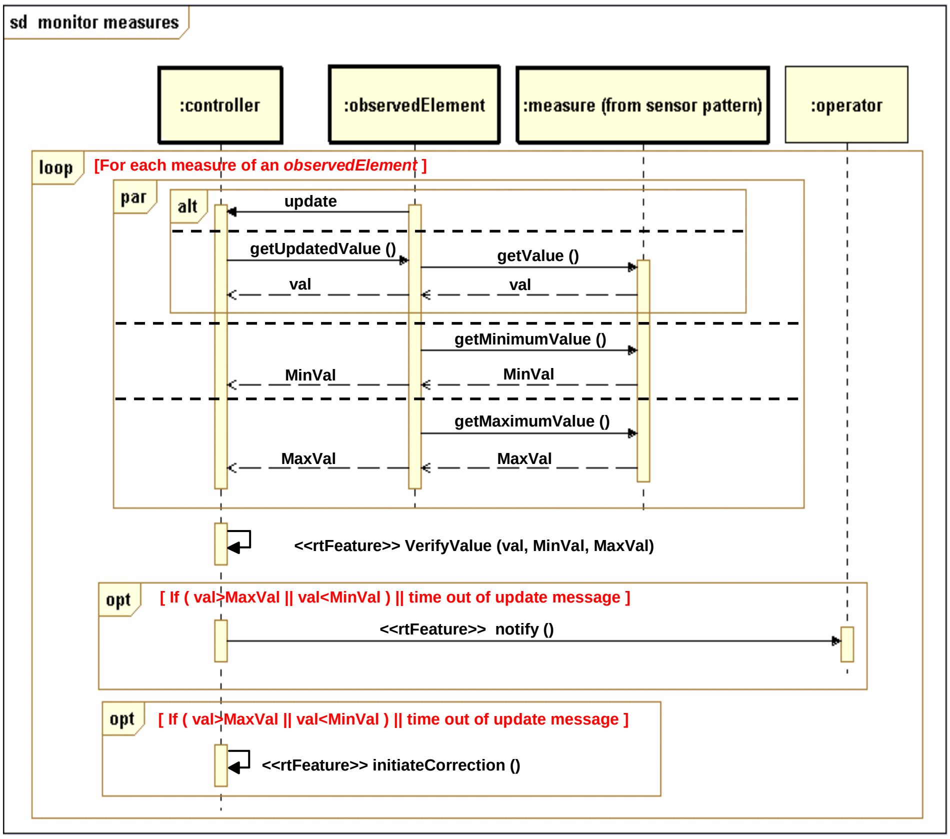
sd monitor measures
:controller
:observedElement
:measure (from sensor pattern)
:operator
loop
[For each measure of an observedElement]
par
alt
update
getUpdatedValue ()
getValue ()
val
val
getMinimumValue ()
MinVal
MinVal
getMaximumValue ()
MaxVal
MaxVal
<<rtFeature>> VerifyValue (val, MinVal, MaxVal)
opt
[If (val>MaxVal || val<MinVal) || time out of update message]
<<rtFeature>> notify ()
opt
[If (val>MaxVal || val<MinVal) || time out of update message]
<<rtFeature>> initiateCorrection ()

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 used to supplement and confirm the data received through the vehicle detector stations and to provide information on local conditions which affect the traffic flow. The processed data are then transmitted at regular time intervals to the Central Computer System to monitor traffic and identify traffic incidents, when they occur.

Figure 4 illustrates the class diagram of the freeway traffic management system reusing RT controller pattern. It indicates that the *controller* monitors two types of elements (*Road_Segment* and *vehicle*). In addition, the *Operator* optional class is instantiated since it is essential to notify the operators of any detected events in the COMPASS system.

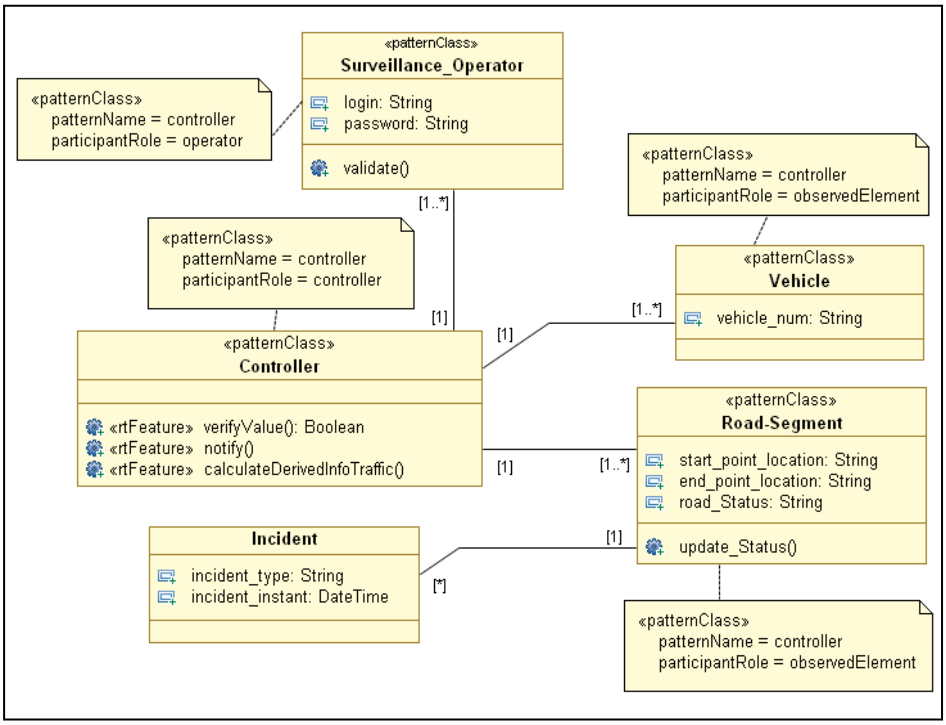

Fig 4. Example of controller pattern instance

5 Conclusion

The design of RT applications differs from the design of classical applications. RT applications have to guarantee that each action (transaction) meets its deadline, and that data are used during their validity interval. Thus, it is necessary (i) to give a great importance to RT applications design and (ii) to benefit from previous experiences of developers by reusing the knowledge previously acquired in the design practices. For this reason, dealing with RT domain engineering becomes a necessity since it allows to identify reusable patterns which reduce the complexity of RT applications design.

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.

16. Yacoub S. M., Ammar H., Pattern-Oriented Analysis and Design: Composing Patterns to Design Software Systems, Published by Addison-Wesley Professional, August 2003.
17. Czarnecki K., Eisenecker U.W., Generative Programming – Methods, Tools, and Applications, Addison-Wesley, 2000.
18. Reinhartz-Berger I., Sturm A., Utilizing domain models for application design and validation, Information and Software Technology, vol 51, pages 1275-1289, 2009.
19. COMPASS Website, Available from: http://www.mto.gov.on.ca/english/traveller/compass/main.htm
20. Douglass B. Real Time UML, Third Edition : Advances in The UML for Real-Time Systems. Pearson Education, Inc, 0-321-16076-2, 2004.
21. Lanusse A., G´erard S., and Terrier F.. Real-time modeling with UML: The ACCORD approach. In J. B´ezivin and P.-A. Muller, editors, The Unified Modeling Language, UML'98- Beyond the Notation. First International Workshop, Mulhouse, France, June 1998, Selected Papers, volume 1618 of LNCS, pages 319–335. Springer, 1999.
22. OMG. ”UML Profile for Schedulability, Performance and Time, v1.1”, formal/2005-01-02, January 2005.
23. Idoudi N., Louati N., Duvallet C., Bouaziz R., Sadeg B. and Gargouri F., How to model a real-time database. Proceedings of 12th IEEE International Symposium on Object-oriented Real-time distributed Computing (IEEE ISORC'2009), Tokyo, Japan, pages 321-325, March 17-20, 2009.
24. Port D., Derivation of Domain Specific Design Patterns. USC Center for software engineering, 1998.