Modeling Real-Time applications with Reusable Design Patterns

Saoussen Rekhis, Nadia Bouassida,
Rafik Bouaziz
*MIRACL-ISIMS, Sfax University, BP
1088, 3018, Sfax, Tunisia.
{saoussen.rekhis,
Raf.bouaziz}@fsegs.rnu.tn
nadia.bouassida@isimsf.rnu.tn*

Claude DUVALLET, Bruno
SADEG
*LITIS, UFR des Sciences et
Techniques, BP 540, 76 058, Le
Havre Cedex, France.
claude.duvallet@univ-lehavre.fr
bruno.sadeg@univ-lehavre.fr*

Abstract

Real-Time (RT) applications, which manipulate important volumes of data, need to be managed with RT databases that deal with time-constrained data and time-constrained transactions. In spite of their numerous advantages, RT databases development remains a complex task, since developers must study many design issues related to the RT domain. In this paper, we tackle this problem by proposing RT design patterns that allow the modeling of structural and behavioral aspects of RT databases. We show how RT design patterns can provide design assistance through architecture reuse of reoccurring design problems. In addition, we present an UML profile that represents patterns and facilitates further their reuse. This profile proposes, on one hand, UML extensions allowing to model the variability of patterns in the RT context and, on another hand, extensions inspired from the MARTE (Modeling and Analysis of Real-Time Embedded systems) profile.

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

1. Introduction

Within the software engineering community, reuse of patterns has long been advocated as an efficient technique to have more profitable and less expensive software applications. With reusable patterns, the design of a new application consists in adapting the existing patterns, instead of modeling one from the beginning.

The need of reuse is confirmed in the field of Real Time (RT) applications which are often considered difficult to design and to implement. In fact, several works have tried to benefit from software reuse in order to develop RT applications. Some works define reusable software components, such as the model of RTCOM components in the ACCORD project [1]. Other works propose RT patterns that provide solutions to recurrent problems of real-time systems (management of resources, distribution, concurrency, and so on) [2]. However, these propositions are not interested in the design of RT databases and the proposed "patterns" remain at a too high level of abstraction to provide for a real design reuse. In fact, a RT database is a database in which both the data and the operations upon the data may have timing constraints [3]. Thereby, its design differs from the design of conventional databases and needs the

capitalization of RT designer's expertise to have high quality and faster software development.

Currently, the demand for RT databases has increased, essentially for applications where it is desirable to execute transactions within their deadlines. Moreover, in order to maximize the number of transactions which meet their deadlines and support freshness of data, several works based on quality of service guarantee, propose to use multi-versions RT data [4] [5]. This reduces data access conflicts between transactions, enhances the concurrency and limits the deadline miss ratio. For this reason, the design of RT databases must support the modeling of multi-versions RT data.

In order to improve and facilitate the RT databases design, we propose in this paper two real-time design patterns. The first is the sensor, which focuses on the design of the generic data stored in RT databases. The second is a modified version of the sensor pattern supporting the multi-versions RT data. These patterns are presented using an UML profile for RT design patterns. This profile offers new stereotypes and expresses the variability and flexibility of the RT domain, in order to be instantiated for various applications. It adapts, also, some MARTE (Modeling and Analysis of Real-Time Embedded systems) [6] profile stereotypes modeling RT aspects at a high abstraction level.

The remainder of this paper is structured as follows. Section 2 presents the related work. Section 3 presents our UML profile which facilitates RT design patterns comprehension and instantiation. Section 4 illustrates the application of this profile through the specification of a RT sensor pattern and a modified version of this pattern that supports multi-version RT data design. Section 5 presents examples of reusing the proposed RT design pattern to model specific applications. Section 6 concludes the paper and gives some perspectives.

2. Related work

Software reuse has long been practiced by software engineers but has traditionally been restricted to the code level: the reuse of individual routines or modules implementing recurring functions. More recently, software engineers have recognized that reuse can take place at a higher abstraction level which is the design level. Design patterns encapsulate reusable design and therefore allow to improve the quality of design.

Works which are interested in developing RT applications with reusable designs, propose patterns intended for real-time systems, called RT patterns. Among these latter, there are the patterns proposed by Douglass [2] and by Schmidt [7]. Douglass proposes architectural patterns, which present solutions to manage concurrency (e.g. *Round Robin pattern, Message Queuing pattern*, etc.), resources (e.g. *Critical Section pattern*, *Priority Inheritance pattern*), distribution (e.g. *Broker pattern*, *Proxy pattern*) and security. He proposes also the mechanistic patterns which refine the architectural patterns and deal with the objects collaborations optimization.

Also within this context, Schmidt [7] defines patterns classified into four categories: the patterns of concurrency for multi-thread systems (e.g. *a thread per object pattern*), the patterns of event (e.g. *a thread per request pattern*, *Reactor pattern and Asynchronous completion token pattern*), the patterns of initialization (e.g. *Acceptor-Connector pattern, Configuration pattern service*, etc.) and finally the patterns of synchronization. These patterns offer solutions to manage concurrency, memory, resources, parallelism, distribution and

security of real-time systems. However, they do not deal with the RT databases modeling problems.

Other works [13] [14] are interested in defining analysis patterns that provide facilities to model functional requirements of RT systems. The analysis patterns proposed by Konard [13] tend to have an inclination to focus primarily on either the structural or behavioral phase of object analysis. They are intended for the embedded systems developers to assist them in defining the relations between the entities of such systems and presenting their behavior in abstract design models. The drawback of these models is that they hide the internal characteristics of the entities, i.e. their attributes and operations. That is, the presented patterns do not assist the RT developers in defining the essential data that must be stored in a RT database. Moreover, they are not intended for the modeling of RT constraints that must be fulfilled by RT data and transactions.

3. The UML profile for RT design patterns

During the specification of RT design patterns, several criteria have to be taken into account: expressivity, variability and definition of constraints. These criteria are considered, in order to have better quality, flexible and more understandable patterns.

In fact, any design language for patterns should be an expressive visual notation based on UML to be easily understood by designers. It should, also, guide the user when adapting a pattern to a specific application. Moreover, it has to express variability in order to determine the variable elements that may differ from one pattern instantiation to another. The correct instantiation of patterns depends on respecting the properties inherent to the solution. These properties are specified by constraints that are generally expressed in OCL (Object Constraint Language) [8].

In the following, we present some UML 2.1.2 [9] basic concepts expressing the variability in the static and behavioral views. Then, we extend this modeling language to specify and instantiate RT design patterns.

In fact, several UML basic concepts express variability in the class diagram (i.e. generalization relationship, constraints interface and template). The generalization relationship represents variation points which are defined by an abstract class and a set of subclasses that constitute the different variants. At least, one of these subclasses is chosen in a pattern instantiation. There are two types of UML constraints that can be applied on the generalization relation:

- {incomplete}: this constraint indicates that the design provides only a sample of subclasses and that the user may add other subclasses in an instantiation.
- {xor}: this constraint indicates that the designer must choose one and only one variant among the presented subclasses during the instantiation.

In the sequence diagram, an interaction sequence can be grouped into an entity, called combined fragment. This latter defines a set of interaction operators, particularly (**alt:** alternative) and (**opt:** optional) operators. The interaction operator (**alt**) indicates that a set of interactions are alternative. It is used with an associated guard that informs the user that only one set of interactions will be chosen. While the interaction operator (**opt**) indicates that a set of interactions represents an optional behavior that can be omitted in a model instance.

Specific domain design pattern are generic designs intended to be specialized and reused by an application. For this reason, we need new notations distinguishing the pattern's common elements which must be kept by any application from the variable elements which change from an application to another. Moreover, when several patterns are instantiated to

design an application, we must differentiate, clearly, among the elements belonging to each design pattern. Thus, we need new concepts for the explicit representation of the pattern elements roles that can assist on the traceability of a pattern.

In the design of a specific domain, the design language must also take into account the specificities of the domain itself. Thereby, the notations used for the representation of patterns intended for RT domain must support the modeling of RT applications characteristics.

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

3.1. Extensions for specifying and instantiating design patterns

We propose new stereotypes distinguishing the optional and fundamental elements participating in a pattern on the one hand, and showing how to compose and to delimit the different patterns in a design of a specific application, on the other hand. Thus, the class and interaction diagrams Meta-models are extended with the stereotypes described in the table 1.

3.2. The profile Metamodel

The design pattern profile metamodel shows the extensions proposed to some meta-classes belonging to the class diagram and interaction diagram metamodels. In order to model RT aspects, the proposed profile imports stereotypes from HLAM (High Level Application Modeling) and NFP (Non Functional Properties) sub-profiles of MARTE [6] (cf. figure 1).

From HLAM sub-profile, we import the <<rtFeature>> stereotype in order to model temporal features. This stereotype extends the meta-classes: 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.

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 pattern profile through the specification of a RT sensor pattern.

Table 1. Proposed stereotypes for specifying and instantiating design patterns

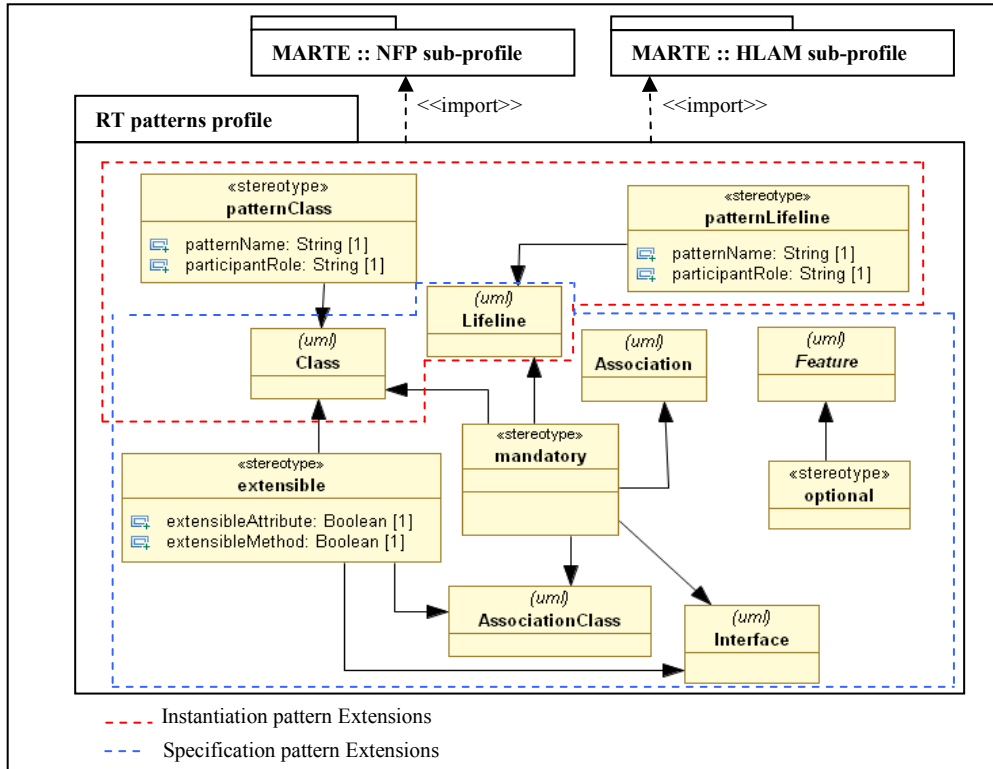|	Stereotype	Signification
Pattern specification	***<<optional>>*** applied to the *Feature* UML Metaclass.	This stereotype is used to specify optional features in UML class diagram. When an attribute (or method) is stereotyped <<optional>>, then it can be omitted in a pattern instance. Each method or attribute which is not stereotyped <<optional>> in a fundamental class means implicitly that it is an essential element, i.e. it plays an important role in the pattern. All the attributes and methods of an optional class are implicitly optional.
	<<mandatory>> applied to the UML Metaclasses: *Class, Association, Interface, Lifeline, ClassAssociation.*	This stereotype is used to specify a fundamental element (association, aggregation,…) that must be instantiated by the designer when he models a specific application. A fundamental element in the pattern is drawn with a highlight line like this class ▭. Each instance of a core class defined in the class diagram is presented with a mandatory lifeline in the interaction diagram. 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.
	<<extensible>> applied to the UML Metaclasses: *Class, Interface and ClassAssociation.*	This stereotype is inspired from {extensible} tagged value proposed in [10]. It indicates that the class interface may be extended by adding new attributes and/or methods. Moreover, we propose to define two properties for the extensible stereotype specifying the type of element (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 (cf. Figure 2, Measure class). Otherwise, this tag takes the value *true*. - *extensibleMethod* tag: It indicates that the designer cannot add new methods when instantiating the pattern if it takes the value *false*. The default value is *true*.
Pattern instantiation	<<*patternClass*>> applied to the *Class* UML metaclass.	Each class, stereotyped <<*patternClass*>>, in a specific application indicates that it is a pattern class. Two properties, relative to this stereotype, are defined: - *patternName* tag : indicates the pattern name, - *participantRole* tag : indicates the role played by the class in a pattern.
	<<*patternLifeline*>> applied to the *Lifeline* UML metaclass	This stereotype is used to distinguish between the objects instantiated from the pattern interaction diagram and those defined by the designer. This stereotype has the same properties than <<patternClass>> stereotype.

Figure 1. RT pattern profile metamodel

4. RT sensor pattern

RT applications that have to be managed by RT databases perform several RT processes. We distinguish among these processes: the RT data acquisition, their control and their RT use. We focus in this paper on modeling data used in the acquisition process through the definition of the RT sensor pattern. This pattern takes into account the acquirement of data from the environment according to two views:

- A static view, which describes the entities, their relationships and the manipulated data that must be stored in the RT database. Each data can be either a classic data or a RT data that has validity interval, beyond which it becomes useless [3].

- A dynamic view, which describes the invocations of methods between the identified entities. Each method execution is considered as a transaction that may be composed of one or many sub-transactions. These methods can be periodic, sporadic or aperiodic. A periodic method updates periodically data acquired from the sensor, called base data [3]. The execution of a periodic method must be achieved before the deadline; otherwise the value to be written will be considered obsolete. A sporadic method updates the derived data that is calculated from base data [3]. Finally, an aperiodic method allows to read/write classical data and to read, only, RT data.

4.1. RT sensor pattern specification

All RT applications depend on the use of sensors to acquire data from the environment. A sensor is defined as a device that measures or detects a physical phenomenon (temperature, pressure, speed, etc.). This detected measure is usable for command ends.

- Interface:
Name: sensor pattern.
Context: Real time applications which need to exploit RT databases.
Intention: The pattern aims to model RT data stored in the RT database and to identify RT constraints related to both: RT data and method that permits their update.

- Solution:
Static specification: Figure 2, presented below, presents the sensor pattern static view.
Participants:
- Sensor: The sensors are classified into passive, active, fixed or mobile sensors. Thus, these types of sensors constitute the variations of the sensor abstract class. In fact, an active sensor takes the transmission initiative of its current value (push mechanism). It must be able to transmit a signal setValue to one object or to a group of objects in order to update the value of a measure. While a passive sensor transmits its value only on the demand of an operator (pull mechanism). It has a method getValue to read the current value. In addition, a mobile sensor allows getting measures at different positions.

- Location: it is an optional class. It can be omitted, when instantiating a pattern, essentially, if the modelled system manages a limited number of fixed sensors and their positions are known to the developer. However, it is important to know the mobile sensor location when acquiring a measure. For this reason, we define an OCL constraint related to the MobileSensor subclass in order to indicate that the designer must instantiate the Location class when he chooses the mobile sensor alternative.

- Measure: this class exists in all RT applications, thus it is a fundamental class drawn with a highlighted line in a RT sensor pattern. It permits to store RT data that are classified into either base data or derived data. Base data are issued from sensors, whereas derived data are calculated from base data. They have the same characteristic of base data (value, timestamp, unit,…). The refreshment of each derived data is required every time one of the base data is updated. In addition, the validity duration of derived data is the intersection of validity duration of every used base data. The relation between base and derived RT data is represented by a reflexive association defined on the Measure class. This association is optional, since it can be omitted in a pattern instantiation, in case the designed application does not have derived measures. However, the association between the Sensor and Measure classes is fundamental because we have to know, for every RT application, the origin of the different values taken from sensors to control the system.

The measure class has an attribute: value, containing the final value captured by the related updateValue () method. It has also an attribute: timestamp, containing the last time at which the measure's value was updated [11] or when this value is produced. The timestamp attribute has DateTime type supported by MARTE profile. It is used to determine whether or not timing constraints have been violated. Moreover, a measure is characterized by a unit and eventually a minimum value and a maximum value that defines the interval for which the system does not detect an anomaly. Each measure is also characterized by the validity duration that represents the time interval during which a measure's value is considered valid. This interval determines, in association with the timestamp attribute value, the absolute consistency of RT data. In fact, the measure's value is considered absolutely consistent

(fresh) with respect to time as long as the age of the data value is within a given interval [11]. The age represents the duration between the timestamp and the current time.
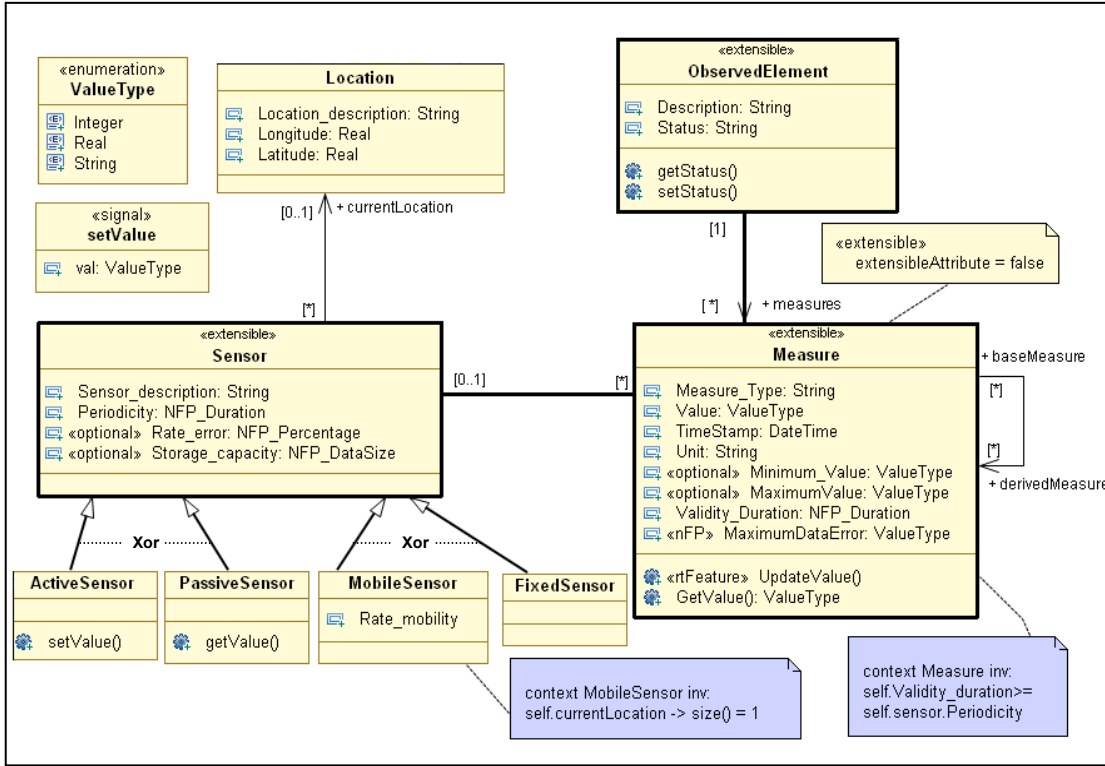

Figure 2. Static specification of the sensor pattern

In addition to the need of fresh data, RT applications have to use precise data in order to reflect the continuous change of the external environment. However, it seems to be difficult for the transactions to both meet their deadlines and to keep the database consistent. For this reason, the quality of data concept is introduced in [5] to indicate that data stored in RT databases may have some deviation from its value in the real world. Thereby, each measure is characterized by the Maximum Data Error (MDE) attribute that represents a non functional property specifying the upper bound of the error. This attribute allows the system to handle the unpredictable workload of the database since an update transaction Tj is discarded if the deviation between the current data value and the updated value by Tj is less or equal to MDE. We propose to associate the <<NFP>> stereotype of MARTE profile to the MDE attribute. This attribute is of the same type as the value attribute.

The attributes (timestamp, validity duration, and maximum data error) defined in this class present the RT data characteristics that must be taken into account in order to support data time semantics and imprecise computations.

- Observed_element: this class represents the description of a physical element that is supervised by one or more sensors. It can be an aircraft, a car, volcanoes phenomenon, and so on. In fact, one or more measure types (i.e. Temperature, Pressure, etc) of each observed element could determinate its evolution.

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

In the dynamic specification of the sensor pattern, we are interested in modeling the RT update transactions and their deadline timing constraints. These transactions are modeled through the invocation and the execution of the *updateValue()* method of the *Measure* class.

Whatever the type of event message of the sensor is (synchronous or asynchronous), the *updateValue()* method allows to change the current value of a measure by considering the result returned by the method *getValue()* or the attribute given as a parameter of the *setValue()* signal.

On the other hand, the stereotype: <<rtFeature>> and the tagged value: occKind, defined in MARTE, are associated to the *updateValue()*method in order to indicate if it is periodic or sporadic. In fact, it is periodic when it sets a base measure's value, whereas it is sporadic when it sets a derived measure's value. Moreover, this stereotype indicates also that the update method has relative and absolute deadlines specified respectively through relDl and absDl tagged values.
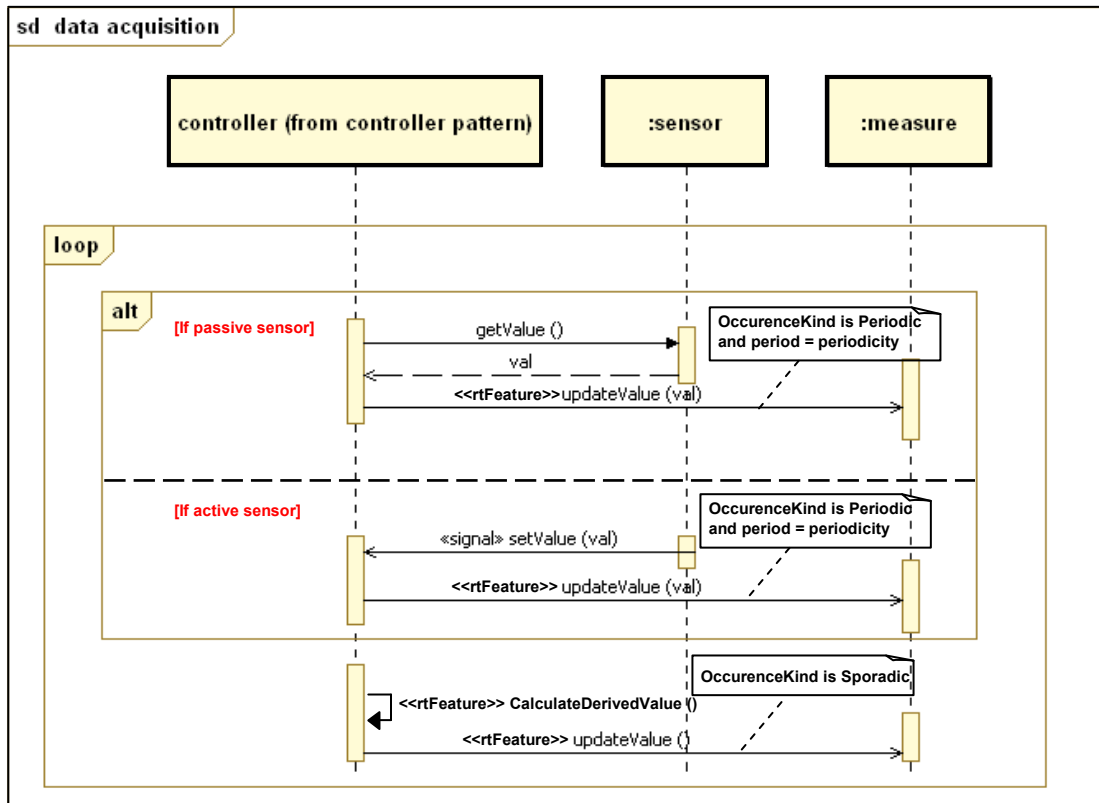

Figure 3. Dynamic specification of sensor pattern

4.2. RT sensor pattern supporting multi-version RT data

In order to preserve data version history, we present in this section a RT sensor pattern supporting multi-versions RT data.

In fact, the multi-versions data allows to maintain for every measure type (velocity, altitude, and so on) multiple versions for a data item. This reduces data access conflicts

between transactions and, then, limits the deadline miss ratio [5]. In fact, most conflict cases come from incompatible accesses when an update transaction wants to modify a data item (i.e. measure's value attribute), accessed by transaction user. The Multi-version technique is used to alleviate this risk through the creation of new versions. However, the number of versions of each RT data is limited. It does not have to exceed a threshold which is a maximum data versions number [5], in order to respect the RT database size.

Table 2 illustrates an example of three versions for the *Speed* measure. The values of the *validity duration* and *Maximum Data Error* attributes are the same for all versions. But, the values of the timestamp attribute changes for each version of speed measure. For this reason, we propose two categories of data which can improve the RT processing: *static data* which does not change during the measurement time and *dynamic data* which represents variable information in time. This classification is carried out according to the evolution of data in time.

To take into account this classification in the sensor pattern, the attributes of *Measure* class must be modelled by two classes: the first one specifies static characteristics of each measure type, whereas the second class stores dynamic RT data acquired from sensors.

Table 2. Example of multi-versions measure.

Speed			
CV (Current Value)	TS (TimeStamp)	VD (Validity Duration)	MDE (Maximum Data Error)
900	10:25:2	6 s	2 s
920	10:25:8	6 s	2 s
925	10:25:14	6 s	2 s

In the following, we present the description of these classes:
- The *Measure-type* class contains the attributes: description, validity duration, maximum data error, minimum value, maximum value and unit of measure. It contains, also, a new attribute which is the maximum data versions number. This latter is related to the non functional requirements specification and is stereotyped with <<Nfp>>. Moreover, we define an OCL constraint relative to the *MeasureType* class and indicating that the number of *Measure* class instances (i.e. the number of version) associated to each *MeasureType* instance must be less than the maximum data versions number.
- *Measure* class contains value and timestamp attributes to take into account the evolution of measure's value and preserve the timestamp of each measure version.

On the other hand, when using a mobile sensor, the position in which each data version is taken, must be stored in the RT database. Thus, the association between the *Measure* and the *Location* classes is compulsory. However, in the case of using a fixed sensor, indicating the location of this sensor is sufficient information. Afterward, all the data versions relative to this sensor are taken in the same location. In this latter case, the association between Sensor and Location classes may be essential when the designed system uses many fixed sensors.

We illustrate in Figure 4 the modifications brought to the sensor pattern class diagram, in order to model the muti-versions RT data.
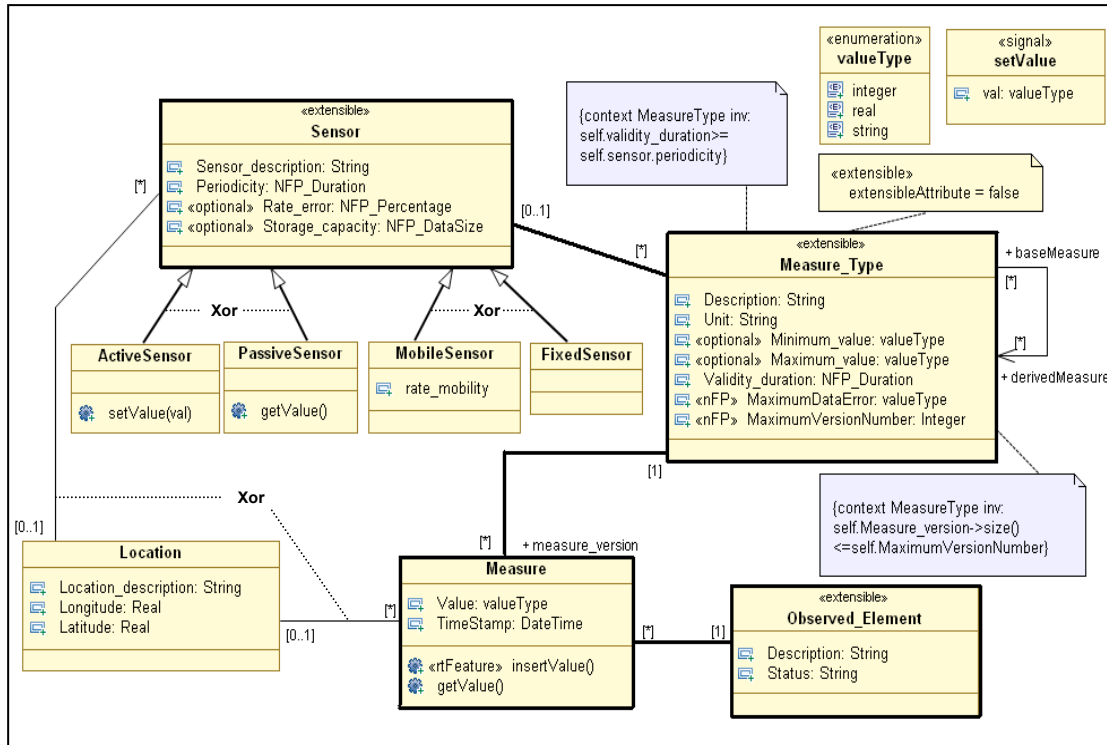
Figure 4. RT sensor pattern supporting multi-versions data

5. RT design patterns instantiation examples

RT design patterns are intended to be instantiated to design specific RT applications. This section proposes two RT applications reusing RT design patterns: the air traffic control system and freeway traffic management system. The first example instantiates the RT sensor pattern and the second reuses the RT sensor pattern that supports multi-version data.

5.1. Air traffic control system example

We present, in this section, the air traffic control example which is an instantiation of the sensor design pattern. The air traffic control application uses a large collection of data describing the aircrafts, their flight plans, and environment data [15]. This includes flight information, such as aircraft identification, altitude, position, speed, origin, destination, route and clearances.

Figure 5 illustrates the static view of the air traffic control application. The design of this application is facilitated by the reuse of the RT sensor pattern. In fact, the designer instantiates first the elements that play a significant role in the sensor pattern (drawn with a highlight line in Figure 2) and substitutes them by specific elements adapted to the context of the air traffic control application. Then, he identifies the variation points presented in the pattern by super classes and chooses the appropriate variants.
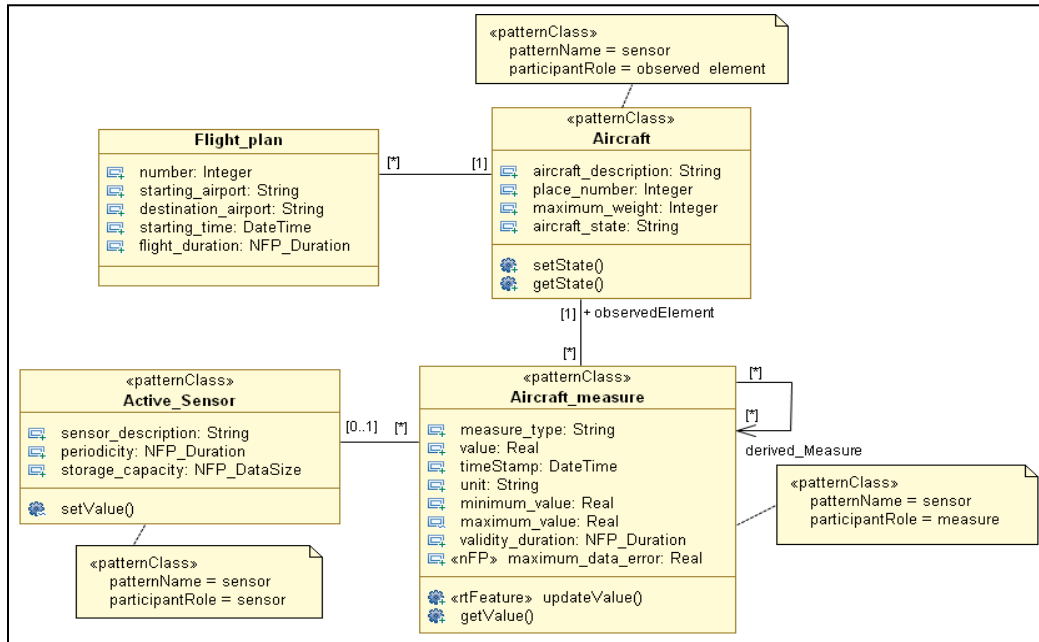
Figure 5. The air traffic control application design

In the case of the air traffic control application, the *Measure* and *Sensor* classes are instantiated respectively by the *Aircraft_Measure* and *Active_Sensor* classes. The Active sensor alternative is chosen because we suppose that all sensors used in this application can publish their values spontaneously. Note that, altitude, speed, location and path constitute instances of *Aircraft_Measure* class. In addition, the *Aircraft* class represents the instance of the *Observed_element* class. After that, the optional elements of the application domain are identified in order to determine those that can be omitted. In our case, the optional reflexive association, modeling the relation between the base and derived measures, is instantiated. In fact, each aircraft has three basic measures which are speed, altitude and location and one derived measure which is aircraft path. The basic measures are periodically updated to reflect the state of an Aircraft. The derived measure is calculated based on altitude and location values, in order to verify if the aircraft deviates from a predetermined path.

Moreover, the pattern name and the role played by each pattern class are indicated by using respectively the tagged values *patternName* and *participantRole* of the stereotype <<patternClass>>. For example, the instantiated class Aircraft_Measure plays the role of a Measure in the Sensor pattern. Thus, the *patternName* tag value is sensor and the *participantRole* tag value is Measure.

Finally, specific elements related to the designed application are added. New attributes (or methods) can be added only for the pattern classes stereotyped <<extensible>>. Notice that in the air traffic control application, instantiating the sensor pattern, some attributes and classes, specific to this application, are added:

- Attributes characterizing the aircraft (e.g., place_number and maximum_weight capacity) since the corresponding observed_element class in the sensor pattern is declared extensible and
- The class "Flight-plan" which indicates the starting and destination airport, the flight duration and so on.

5.2. Freeway traffic management system example

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 [12]. We focus precisely on modeling the compass acquisition data subsystem and we explain how this design issue can be facilitated by the reuse of the RT sensor pattern supporting multi-versions data. This pattern is chosen since the COMPASS system stores historical traffic data at different times, for retrieval and analysis purposes.

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. These processed data are then transmitted at regular time intervals to the Central Computer System. 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 computer system uses the acquired data stored in a real time database to monitor traffic and identify traffic incidents, when they occur.

Figure 6 illustrates the class diagram of the COMPASS system reusing RT sensor pattern supporting multi-version data.

First, the fundamental elements of the pattern are instantiated. Thus, the *Measure_Type, Measure*, *Sensor* and *Observed_element* classes are instantiated respectively by *InfoTraffic_MeasureType, InfoTraffic, ActiveSensor, Vehicle* and *RoadSegment* classes of the freeway traffic management system. The vehicles and road segments represent the physical elements that are supervised by sensors. Moreover, the sensors used in the COMPASS system can publish their acquired data spontaneously every twenty seconds. For this reason, the active sensor alternative is chosen.

For each measure taken from the environment of this system and stored in the database, the designer must specify the value, the timestamp and the validity interval to verify the temporal consistency of traffic collected data. For example, the value of the vehicle speed measure is temporally consistent as long as it is no more than twenty seconds. In addition, the designer must specify the minimum and maximum thresholds of each taken measure in order to determine the abnormal values for which the COMPASS system may detect an incident. Thereby, vehicle speed, vehicle length, traffic volume and occupancy constitute the instances of *InfoTraffic_MeasureType* class. The value evolution of each measure type is stored in the *InfoTraffic* class. The speed and length measures are relative to the *Vehicle* class. Whereas, the traffic density and occupancy measures are relative to the *RoadSegment* class.

Finally, specific elements relative to the application domain are added to the design. In our case, some elements specific to the freeway traffic management application, are added:

- a composition relation between the *ActiveSensor* class and *RoadSegment* class in order to determine the Vehicle detector stations and supervision cameras placed in each road segment and

- a *RoadLink* class with a num_link attribute. This class indicates the segments that compose each road link through the composition relation defined between *RoadSegment* and *RoadLink* classes.
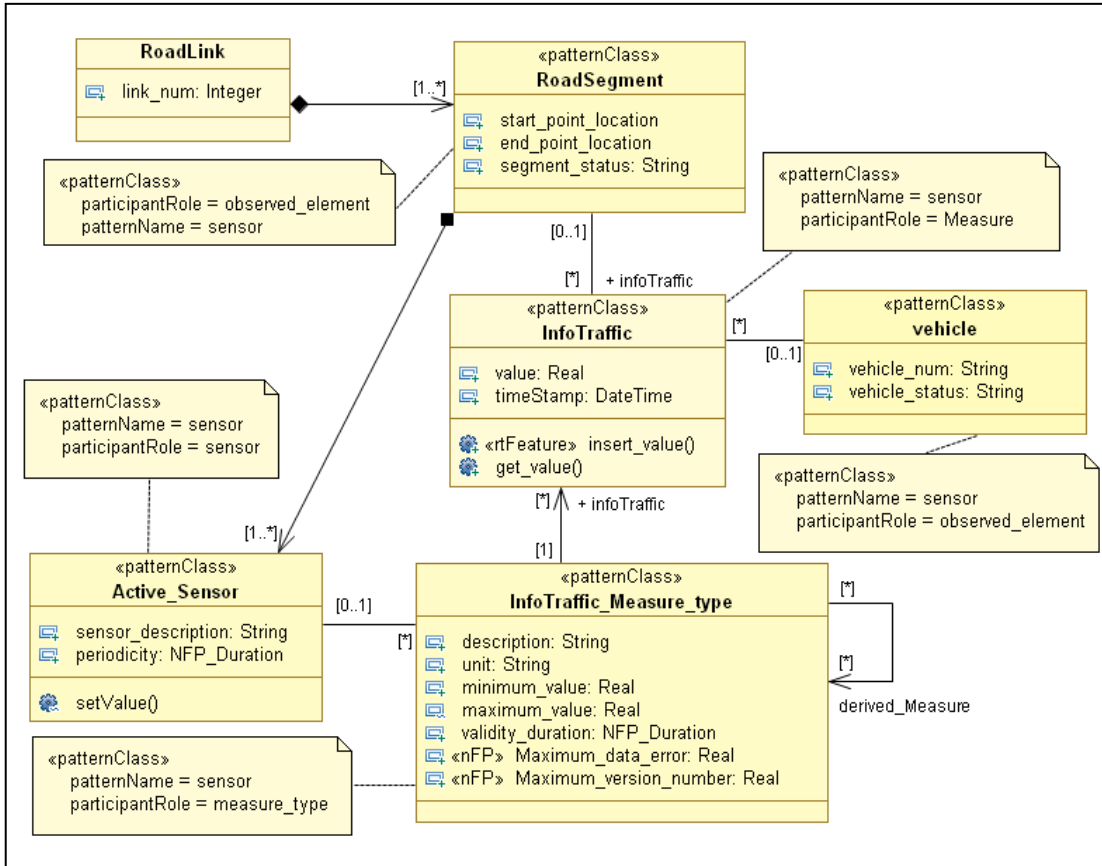

Figure 6.   The freeway traffic management application design

6. Conclusion

The reuse technique allows to capitalize the knowledge of the experts and to reduce software development complexity. In this paper, we proposed an approach based on reusable design patterns to reduce the complexity of RT databases modeling. These patterns help designers to develop RT applications expressing time-constrained data and time-constrained methods.

Besides, we propose UML-based extensions expressing the variability and tracing design patterns. This leads to have common standard notations for defining RT design patterns. This allows patterns to be exchanged among designers in a more readily manner, consequently to improve RT design models.

Our future works include: 1) the definition of additional patterns in order to model other aspects of RT databases; 2) the integration of the design patterns in the context of 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] Tesanovic J. A., Nystrom D. and Norstrom C., Towards aspectual component-based development of real-time systems. Proceedings of the 9th Inter. Conf. of Real-Time and Embedded Computing Systems and Applications (RTCSA'03), pp. 558-577, 2003.

[2] Douglass B. P., Real-Time Design Patterns: Robust Scalable Architecture for Real Time Systems, Addison-Wesley Edition, September 27, 2002.

[3] Ramamritham K., Son S., and DiPippo L., Real-Time Databases and Data Services. Real-Time Systems, pp. 179–215, 2004.

[4] Bouazizi E., duvallet C., Sadeg B., Multi-Versions Data for improvement of QoS in RTDBS, Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05), 2005.

[5] Amirijoo M., Hansson J., and Son S. H., Specification and management of QoS in real-time databases supporting imprecise computations. IEEE Transactions on Computers, 55(3), 2006.

[6] OMG, A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, OMG document number: ptc/2008-06-09, 2008.

[7] Schmidt D. C., Stal M., Rohnert H. and Buschmann F., Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Wiley & Sons, 2000.

[8] OMG, UML 2.0 OCL specification, 2003.

[9] OMG, Unified Modeling Language (UML) Infrastructure, v2.1.2, formal/2007-11-04, November 2007.

[10] Bouassida N., Ben-Abdallah H., Extending UML to guide design pattern reuse, Sixth Arab International Conference On Computer Science Applications, Dubai, 2006.

[11] Ramamritham K., Real-Time Databases. Journal of Distributed and Parallel Databases, 1(2):199–226, 1993.

[12] COMPASS Website, Available from: http://www.mto.gov.on.ca/english/traveller/compass/main.htm.

[13] Konard S.J., Cheng B H.C. and Campbell L. A., "Object Analysis Patterns for Embedded Systems", IEEE Transactions on Software Engineering, Vol. 30, No. 12, December 2004.

[14] Jawawi D., Deris S., and Mamat R.. Software Reuse for Mobile Robot Applications Through Analysis Patterns, The International Arab Journal of Information Technology, Vol. 4, No. 3, 2007.

[15] Locke D., Applications and system characteristics. In Real-Time Database Systems: Architecture and Techniques, pages 17–26. Kluwer Academic Publishers, 2001.