How to Model a Real-Time Database?

Conference Paper · March 2009

DOI: 10.1109/ISORC.2009.17 · Source: DBLP

5 authors, including:

Claude Duvallet
Université du Havre
86 PUBLICATIONS **249** CITATIONS
SEE PROFILE

Rafik Bouaziz
University of Sfax
226 PUBLICATIONS **812** CITATIONS
SEE PROFILE

Bruno Sadeg
Université du Havre
86 PUBLICATIONS **285** CITATIONS
SEE PROFILE

Faiez Gargouri
University of Sfax
434 PUBLICATIONS **1,560** CITATIONS
SEE PROFILE

How to model a real-time database?[*]

Nizar Idoudi, Claude Duvallet and Bruno Sadeg
UFR des Sciences et Techniques
25 rue Philippe Lebon, BP 540, 76 058 Le Havre Cedex, FRANCE
{nizar.idoudi, claude.duvallet, bruno.sadeg}@univ-lehavre.fr

Nada Louati, Rafik Bouaziz and Faiez Gargouri
MIRACL-ISIMS, BP 1030, 3018 Sfax, TUNISIE
{Raf.Bouaziz, Faiez.Gargouri}@fsegs.rnu.tn

Abstract

Real-time databases deal with time-constrained data and time-constrained transactions. The design of this kind of databases requires the introduction of new concepts to support both data structures and the dynamic behaviour of the database. In this paper, we give an overview about different aspects of real-time databases and we clarify requirements of their modeling. Then, we present a framework for real-time database design and describe its fundamental operations. A case study demonstrates the validity of the structural model and illustrates SQL queries and Java code generated from the classes of the model.

1. Introduction

Real-time databases have to deal with time-constrained data and time-constrained transactions. They are now being used for several applications such as space project control, process control, financial market and air traffic control systems. In each of these time-critical applications, data about the target environment must be continuously collected from the real-world and processed in a timely manner to generate real-time responses. A real-time database has two distinguishing features: the notion of temporal consistent data, and the ability to place real-time constraints on transactions. Some of its data must not only be logically consistent, but also temporally consistent, i.e., they must closely reflect the current state of the controlled environment. However, data are collected at discreet moments. Hence they often represent an approximation of the reality. As time advances continually, a real-time data value becomes less and less accurate, until the moment where it does not reflect any more the state of the environment. At this time point, we say that this data value is temporally inconsistent. Temporal consistency can be measured in two ways: *absolute consistency* and *relative consistency* [16]. A data item is considered absolutely consistent if and only if its age is within a specified time interval. It should be often updated. The age represents the time duration between its timestamp and the current time. For example, the speed age value must not exceed five seconds; it verifies its absolute consistency constraint as long as it is no more than five seconds old. Relative consistency concerns data derived from other ones. For example, the lane of an aircraft is derived from the location and the altitude data items. Its temporal consistency depends of location and altitude data consistency. So, real-time data are subdivided into two types: *sensor data* and *derived data* [18]. Sensor data are data issued from sensors. Derived data are data computed using sensor data (e.g., lane data).

Transactions in a real-time database environment are also subdivided into two classes: *update transactions* and *user transactions* [16]. Update transactions are used to update the values of real-time data in order to reflect the state of the real world. They are executed periodically to update sensor data, or sporadically to update derived data [8]. User transactions, representing user requests, arrive aperiodically. They may read or write non real-time data, but only read real-time data.

Real-time databases are therefore specific. Their design need appropriate concepts and tools which are not available under systemic or object oriented methods. UML, the most used modeling language nowadays, can not, in its standard form, satisfy the requirements of such design. Indeed, it is a general language used to model object-oriented applications across a wide range of domains. Nevertheless, its extension mechanism, based on the concepts of profile and stereotype, allows it, if enhanced, to support new concepts and tools, and to become suitable for a particular domain

[*]This work is supported by the Haute-Normandy region **CPER** project "*Logistic Transport Network and Information Technics*".

needing specific software engineering activities. Recently, an UML profile for Modelling and Analysis of Real-Time and Embedded systems (MARTE) has been standardized by the OMG [12] [3]. However, the design of real-time databases differs from the design of conventional real-time systems. The designers of real-time databases must consider both temporal aspects of data and timing constraints of transactions [20] [8]. The design of this kind of databases is thus performance-and semantic-dependent. It must consider factors such as sensor data, derived data and Quality of Data (QoD) management, temporal semantics in transaction scheduling algorithms, concurrency control protocols, disk caching, and buffer management protocols to meet the timing constraints defined by real-time applications. Then, which concepts and tools are suitable to define the design of real-time databases? How to define and to implement an UML profile supporting these requirements? How to implement a real-time database model under a technical environment? This paper deals with these issues and aims to bring contributions in real-time databases design.

The paper is structured as follows. Section 2 briefly covers related works. Section 3 describes our real-time object-oriented data model. Section 4 presents a set of stereotypes that allows our UML-RTDB profile to express real-time database features in a structural model. Section 5 quotes different mapping rules of an UML-RTDB diagram to an object relationship schema. Finally, the last section draws some conclusions.

2. Related work

As the need for using real-time database grows, there is a need to define an UML profile supporting real-time database requirements. Several UML approaches were already proposed to take into account the real-time system requirements, such as *UML-RT* [19], *RT-UML* [5], *UML-SDL* [9] and *ACCORD/UML* [10]. For example, the basic concepts of *RT-UML* were integrated in the standard UML through the UML profile for Schedulability, Performance, and Time (denoted SPT profile) [13], which is recently replaced by the UML profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [12]. However, UML constructs used by these approaches do not support real-time database requirements. A real-time database is a database in which both the data and the operations upon the data may have timing constraints [16]. In fact, real-time databases have all requirements of traditional databases, such as the management of accesses to structured, shared and consistent data, but they also require management of time-constrained data and time-constrained transactions [2].

To the best of our knowledge, there is only one UML-based proposal for real-time databases modeling [4]. In their work, the authors have defined an UML package for specifying RTSORAC [1] [21] object, called *RT-Object*. However, the RT-Object package is based on the Extension Mechanisms package of UML1.3 which is a past standard. Furthermore, imprecise computation encapsulated within the RTSORAC object is defined in the context of *Epsilon Serializability* on transactions [17], and does not support the notion of *QoD* introduced in [1]. The *QoD* concept allows a robust and controlled behavior of real-time databases during transient overloads, based on *Feedback Control Real-Time Scheduling* [11].

The framework proposed in this paper supplies, like RT-Object, concepts and tools for real-time database modeling. But, unlike RT-Object, UML-RTDB, the profile of this framework, supports the QoD concept that we define for real-time attributes [8], on one hand, and it contains a set of stereotypes to express dynamic semantics of real-time attributes and real-time object features, on the other hand. These stereotypes are defined under UML.2.1.2 Profiles package [15]. In addition, UML-RTDB allows to specify two kinds of real-time attributes, *sensor* attributes and *derived* attributes, in order to satisfy the current real-time applications requirements.

3. Real-time object-oriented data model

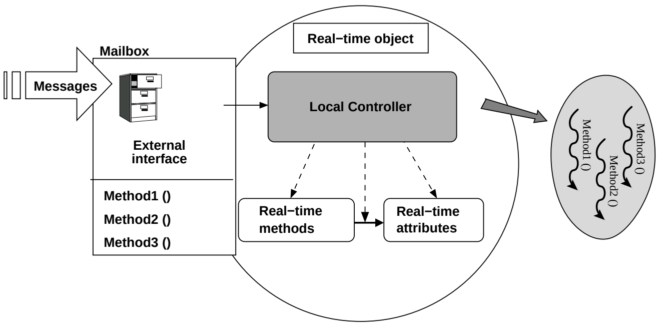


Figure 1. Aircraft object.

Our work is based on a particular object model, named *real-time object-oriented data model*, on which we incorporate time-constrained data and time-constrained transactions of real-time databases [8]. Thus, a real-time database is a collection of real-time objects (RTO) which are used to manage time-critical dynamic systems in the real world. Real-time objects are the real-time object-oriented database entities. They represent dynamic entities of time-critical dynamic applications in the real world [7]. We define a real-time object as an extension of the real-time object, as used in the ACCORD/UML approach [6]. It encapsulates

[1]RTSORAC: Real-Time Semantic Objects Relationships And Constraints

time-constrained data, time-constrained methods and concurrency control mechanisms. As shown in figure 1, each real-time object is made of four components: (i) a set of real-time attributes, (ii) a set of real-time methods, (iii) a mailbox, and (iv) a local controller [7] [8].

4. The UML-RTDB profile

In this section, we present an UML profile, entitled UML-RTDB, which is a specialized variant of the UML2.1.2 for real-time database applications. The main aim of our proposal is to supply to the designers of real-time databases UML extensions to support real-time database requirements. This UML extension is specified in the UML metamodel by a stereotype. The stereotype define how an existing metaclass may be extended, and enables the use of platform or domain specific terminology or notation, in addition to the ones used for the extended metaclass [15]. In our work, UML-RTDB stereotypes extend metamodel classes with specific sensor and derived attributes, specific periodic and sporadic operations and a specific real-time class that allow the design of class diagrams for real-time databases. We base our proposal on the *Extension relationship* proposed in UML2.1.2 *Profiles* package [15].

4.1. Real-time data type

As defined in [8], each real-time attribute value is characterized by a timestamp, which indicates the time at which it was last updated. So, for each real-time attribute value corresponds a timestamp, which distinguishes it from other attribute's values. Thereby, as illustrated in figure 2 for the attribute *Speed*, the values of the *VD* and *MDE* fields are the same for all real-time attributes. But, the values of the ***TS*** field change for each real-time attribute.

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

Figure 2. Illustration of the dynamic semantic of a real-time attribute.

For this reason, we define three new UML data types, called *RTInteger*, *RTReal* and *RTString*, that describe the type of real-time attribute values. As shown in figure 3, these metaclasses are characterized by two properties:

- Value: it indicates the attribute value written by the related update method. It is of the type *Integer*, *Real* or *String* in the case of the metaclass *RTInteger*, *RTReal* or *RTString*, respectively.

- Timestamp: it indicates the last time at which the attribute's value was updated.

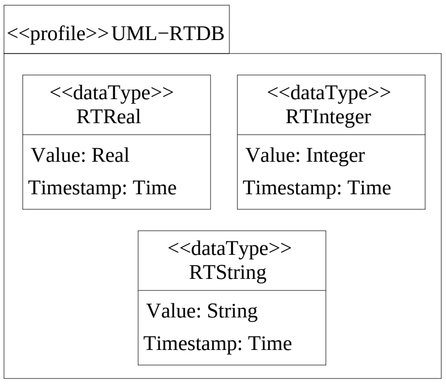


Figure 3. Data types of UML-RTDB profile.

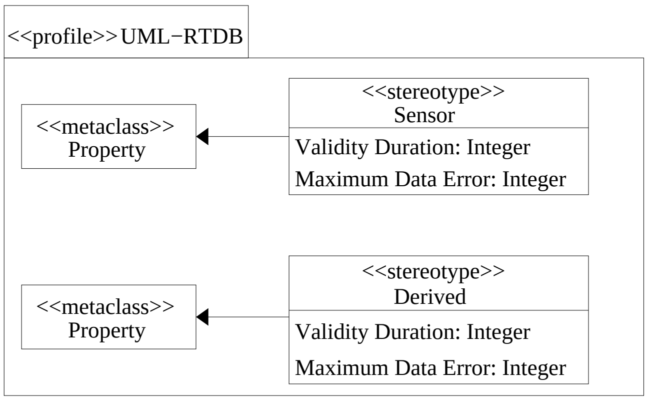
4.2. Sensor and Derived stereotypes

A real-time attribute may be either *sensor* or *derived* [8]. To the best of our knowledge, there is no work which characterizes derived data by a *Maximum Data Error* (MDE). However, since derived data are the data calculated from sensor data which have already some deviation from their values in the real world (MDE), then derived data have automatically some deviation from their values in the real world (MDE). For example, a data read from a sensor has an MDE of 1m. If a derived attribute is based on the formula 2 * the sensor data, it would then have an MDE of 2m. For this reason, in our work, we characterize derived data by an MDE. In the general case derivation of the MDE value would not be as simple, but it remains an interesting and innovative perspective of research.

As shown in figure 4, we define two stereotypes, ≪*Sensor*≫ and ≪*Derived*≫, to declare respectively sensor attributes and derived attributes in the class diagrams. Each of these stereotypes is characterized by two properties:

- Validity duration: it indicates the amount of time during which the attribute value is considered valid. It is of the type *Integer*.

- Maximum data error: it indicates the maximum deviation tolerated between the current attribute value and the updated value. It is of the type *Integer*.



<<profile>>UML−RTDB
<<metaclass>> Property
<<stereotype>> Sensor
Validity Duration: Integer
Maximum Data Error: Integer
<<metaclass>> Property
<<stereotype>> Derived
Validity Duration: Integer
Maximum Data Error: Integer

Figure 4. Definition of Sensor and Derived stereotypes.

4.3. Periodic and Sporadic Stereotypes

We characterize the UML-RTDB profile by three stereotypes, ≪*Periodic*≫, ≪*Sporadic*≫ and ≪*Aperiodic*≫, to declare respectively periodic methods, sporadic methods and aperiodic methods [8] in the class diagrams. As illustrated in figure 5, we define an abstract stereotype, named ≪*Update*≫, that generalizes these latter stereotypes. It is characterized by a *Deadline*, which indicates the last time by which the method execution must be completed. In addition, we characterize the ≪*Periodic*≫ stereotype by a *Period* in order to define the periodicity of the methods.

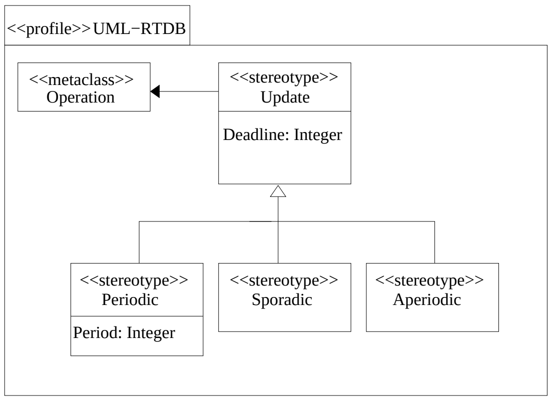


Figure 5. Definition of Periodic and Sporadic stereotypes.

4.4. Real-Time Class Stereotype

A real-time database may be viewed as an augmented database system. It then has queries, schemas, transactions, commit protocols, concurrency control support, and storage management [20]. So, the design of a real-time database has to take into account the management of all these components. That's why we define a ≪*RealTimeClass*≫ stereotype (cf. Figure 8) in order to deal with the time-constrained data, time-constrained operations, parallelism, and concurrency property inherent to real-time databases. The ≪*RealTimeClass*≫ stereotype is added to classes in order to specify that their instances will encapsulate real-time data, real-time operations, and a local concurrency control mechanism.

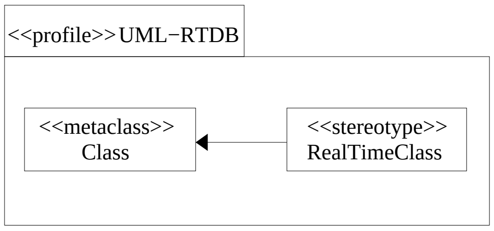


Figure 6. Definition of RealTimeClass stereotype.

5. From an UML-RTDB diagram to an Object Relationship schema

Many development tools are based on relational databases [2]. Although the relational model is useful for many applications, we believe that it is not as well-suited as an object-oriented database model for applications that require complex data management, have complex relationships between data, a suitable support for timing constraints, and more scheduling flexibility than serialisability can provide. In the relational model, the mapping of an UML-RTDB class generates a large number of tables. So, queries need many joins to retrieve needed data, and their execution becomes expensive (i.e. a large amount of time). The main reason of this inadequacy is that relational model can deal only with simple data. In other words, relational model use a tabular representation of the real-word entities. For instance, the mapping of the *Aircraft* class gives seven tables, one to represent the atomic attributes of the *Aircraft*, and six tables representing every object connected with the *Aircraft*: *Direction*, *Location*, *Altitude*, *Speed*, *Path* and *Lane*. Each table contains a *foreign key* referring to the *primary key* of the *Aircraft*. The selection of all information of an *Aircraft* needs six join operations. This decreases significantly the performance of the system.

For these reasons, we see that the mapping of real-time class to the object-relational model is more suitable than the relational model. In fact, the object-relational model allows the use of simple or complex structures. Each structure is defined through an appropriate mechanism, called **U**ser **D**efined **T**ype (*UDT*). An object-relational table is thus defined by means of either complex or simple data. The recordings of an object-relational table represent concret objects, which have methods endowed with an **O**bject **Id**entity (*OID*).

Moreover, object-relational technology is a relational technology which is extended with new capabilities, such as *methods*, *UDT*, etc. It offers two advantages: firstly, it is compatible with relational technology and provides a better support for complex object. Secondly, object-relational databases are becoming commonplace because many commercial **D**ata**B**ase **M**anagement **S**ystems (*DBMS*) add object-oriented capabilities to their functionalities, such as *Oracle 11g*, *IBM DB2* and *PostgreSQL*. For These reasons, we base our work on the object-relational database design.

To map real-time classes into the object-relational model, we proceed in the following manner.

5.1. Mapping of derived attributes

Derived attributes are mapped through the following actions:

Action 1: Creation of an **U**ser **D**efined **T**ype (*UDT*), named *RealTime*, which contains three fields *CV*, *TS* and *VD* (cf. Query 1).

Action 2: For every attribute whose multiplicity is greater than one, we proceed in the following manner:

1. We create an *UDT* which represents a nested table, named *NT_RealTime*, of *RealTime* type, when the exact value of the multiplicity is not mentioned.

2. We create an *UDT* which represents an array, named *ARR_RealTime*, of *RealTime* type, when the exact value of the multiplicity is mentioned.

Query 1 Creation of an *UDT* for derived attributes

```
SQL create type RealTime as object
      (Value number,
      TimeStampValue timestamp,
      ValidityDuration number)
      not final;
      /
```

5.2. Mapping of sensor attributes

Sensor attributes are mapped through the following actions:

Action 1: Creation of an *UDT*, named *RealTimeSensor*, composed of four fields: *CV*, *TS*, *VD* and *MDE* (cf. Query 2).

Action 2: Is the same than action 2 for the derived attribute mapping (cf. section 5.1).

Query 2 Creation of an *UDT* for sensor attributes

```
SQL create type RealTimeSensor under RealTime
      (MaximumDataError number)
      /
```

5.3. Mapping of methods

When defining an *UDT*, we can specify all the methods which allow us to manipulate the real-time object characteritics. The methods can be programmed using *functions* and *procedures* of an *UDT*. Then *triggers*, *functions*, *procedures*, *packages*, *constraints* and *privileges* allow to execute the encapsulation aspects which are not automatically taken into account, as the *visibility rules* of an attribute or a method. These aspects can be handled by object-relational views.

5.4. Mapping of real-time classes

Real-time class is mapped through the following actions:

Action 1: Creation of an *UDT*, named *NOM_CTR_TYPE*, which contains the following fields (cf. Query 3):

Query 3 Creation of an *UDT* for aircraft class

```
SQL create type Aircraft as object
      (Identifier varchar2(15),
      Destination varchar2(15),
      Direction RealTimeSensor,
      Location RealTimeSensor,
      Path RealTime,
      Lane RealTime)
      /
```

1. Classical attributes of the real-time class.

2. Sensor attributes of the real-time class, with the suitable types (types obtained from the mapping of sensor attributes: *AN_TYPE*, *NT_AN_TYPE* or *ARR_AN_TYPE*).

3. Derived attributes of the real-time class, with the appropriate types (types obtained from the mapping of the derived attributes: *AN_TYPE*, *NT_AN_TYPE* or *ARR_AN_TYPE*).

Action 2: Creation of an object-relational table, that has the same name as the real-time class. Then, we add other constraints (*primary key*, *foreign key*, etc.) (cf. Query 3).

Figure 7 illustrates the structure of an *Aircraft* real-time class. It encapsulates classical attributes, and real-time sensor and derived attributes.

Aircraft−Id	Direction				Location				Path			Lane		
	Current Value	TimeStamp	Validity Duration	Maximum Data Error	Current Value	TimeStamp	Validity Duration	Maximum Data Error	Current Value	TimeStamp	Validity Duration	Current Value	TimeStamp	Validity Duration

Figure 7. Structure of an Aircraft real-time class.

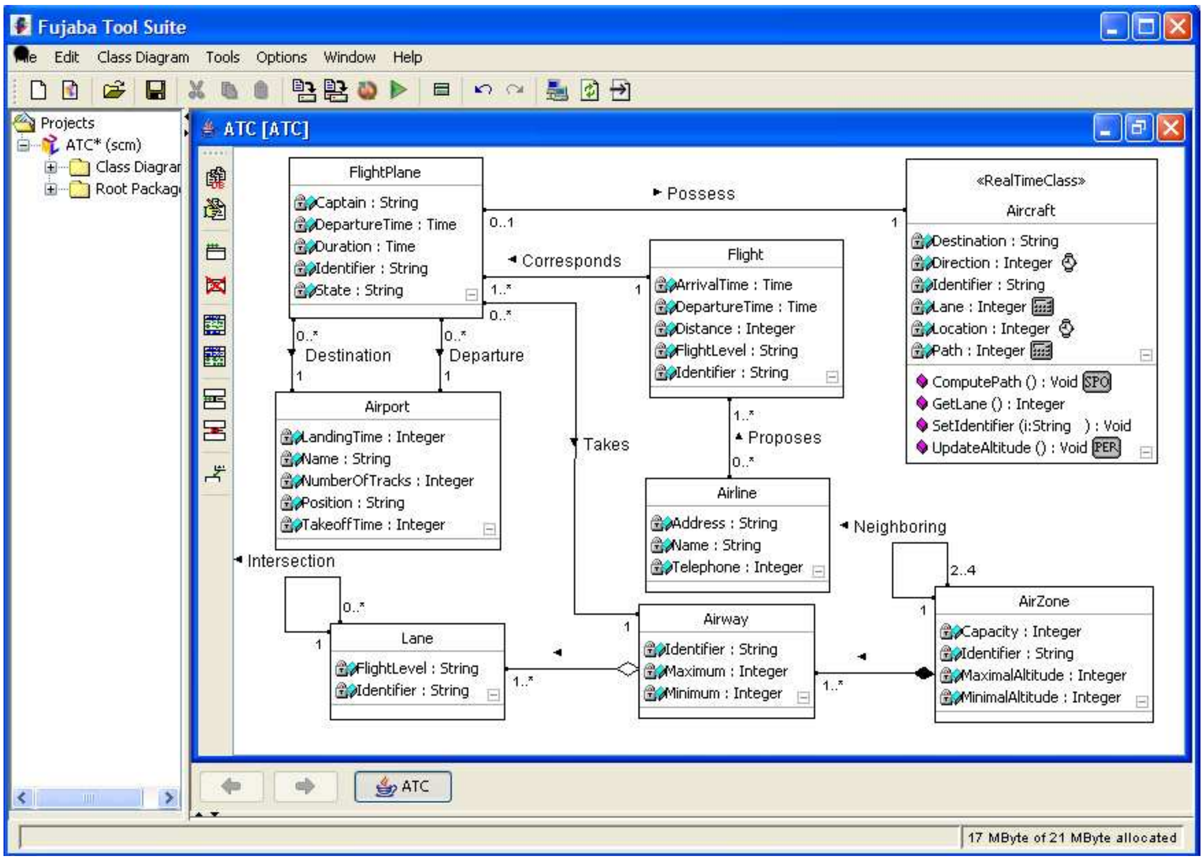


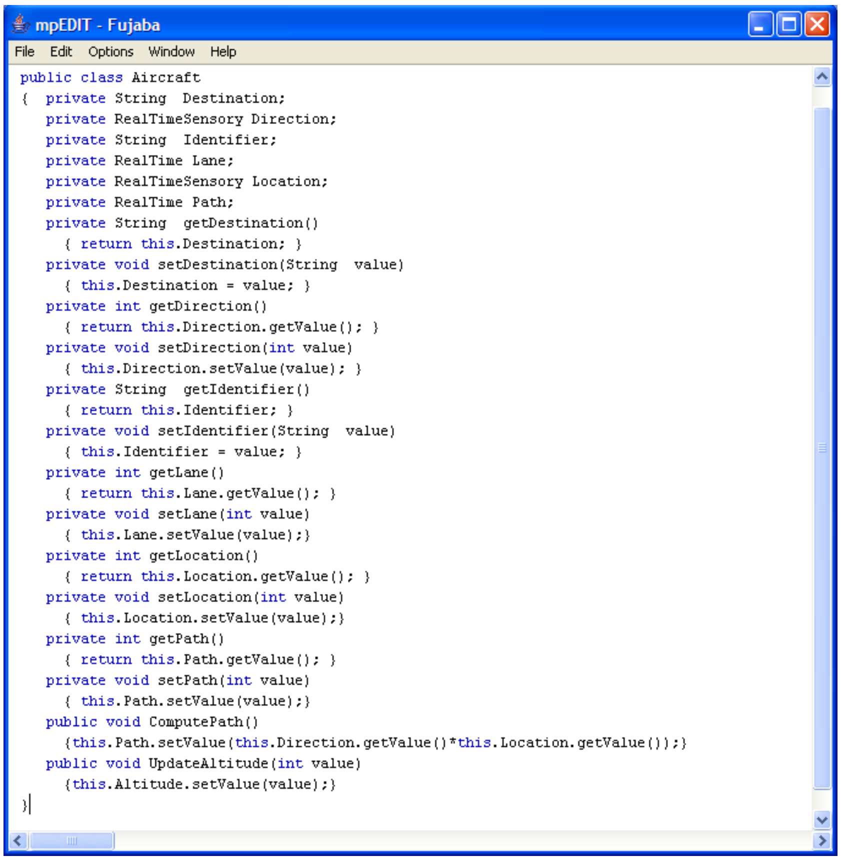
Figure 8. Air traffic control class diagram.

6. Implementation

The implementation of our UML-RTDB components is done through an extension of an UML CASE Tool, named Fujaba (**F**rom **U**ML to **J**ava **A**nd **B**ack **A**gain). The Fujaba environment aims to provide round-trip engineering support for UML and Java. The main distinction to other UML tools is its tight integration of UML class and UML behaviour diagrams to a visual programming language. This integration enables Fujaba to perform a lot of static analysis work, facilitating the creation of a consistent overall specification. In addition, it turns these UML diagrams into a powerful visual programming language and allows covering the generation of complete application code. Since Fujaba is open source, it will be possible to add to it the appropriate tools and make it able to accept real-time database specification. Figure 8 shows the class diagram under Fujaba of our air traffic control application. We have chosen a "*Watch*" icon to indicate sensor attribute and a "*Calculator*" icon to indicate derived attribute of Aircraft real-time class. In addition, a "*SPO*" icon is used to indicate sporadic methods and "*PER*" icon to indicate periodic methods. Figure 9 presents Java code generated from the Aircraft real-time class. Figure 10 presents SQL queries also generated from the Aircraft real-time class.

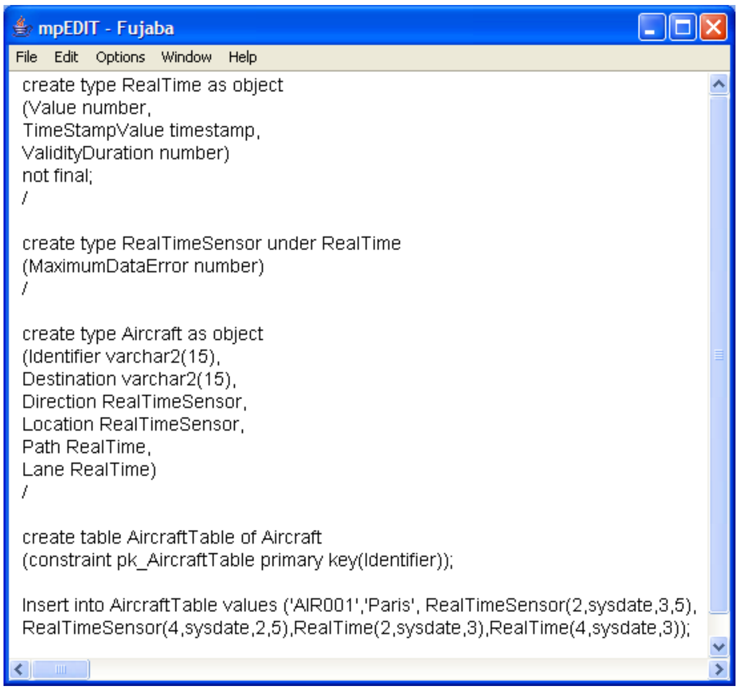
7. Conclusion and future work

Many real-time applications need a database environment. But classical databases can not satisfy all requirements of these applications. We have studied, in this paper, the temporal requirements of real-time databases, for data and operations. We have proposed, to specify dynamic semantics and complex data structure of these databases, a set of appropriate concepts and tools, giving a real-time object-oriented data model. This model incorporates new



```
public class Aircraft
{  private String  Destination;
   private RealTimeSensory Direction;
   private String  Identifier;
   private RealTime Lane;
   private RealTimeSensory Location;
   private RealTime Path;
   private String  getDestination()
     { return this.Destination; }
   private void setDestination(String  value)
     { this.Destination = value; }
   private int getDirection()
     { return this.Direction.getValue(); }
   private void setDirection(int value)
     { this.Direction.setValue(value); }
   private String  getIdentifier()
     { return this.Identifier; }
   private void setIdentifier(String  value)
     { this.Identifier = value; }
   private int getLane()
     { return this.Lane.getValue(); }
   private void setLane(int value)
     { this.Lane.setValue(value);}
   private int getLocation()
     { return this.Location.getValue(); }
   private void setLocation(int value)
     { this.Location.setValue(value);}
   private int getPath()
     { return this.Path.getValue(); }
   private void setPath(int value)
     { this.Path.setValue(value);}
   public void ComputePath()
     {this.Path.setValue(this.Direction.getValue()*this.Location.getValue());}
   public void UpdateAltitude(int value)
     {this.Altitude.setValue(value);}
}
```

Figure 9. Java code generation from Aircraft real-time class.



```
create type RealTime as object
(Value number,
TimeStampValue timestamp,
ValidityDuration number)
not final;
/

create type RealTimeSensor under RealTime
(MaximumDataError number)
/

create type Aircraft as object
(Identifier varchar2(15),
Destination varchar2(15),
Direction RealTimeSensor,
Location RealTimeSensor,
Path RealTime,
Lane RealTime)
/

create table AircraftTable of Aircraft
(constraint pk_AircraftTable primary key(Identifier));

Insert into AircraftTable values ('AIR001','Paris', RealTimeSensor(2,sysdate,3,5),
RealTimeSensor(4,sysdate,2,5),RealTime(2,sysdate,3),RealTime(4,sysdate,3));
```

Figure 10. Generation of SQL queries from an Aircraft real-time class.

types of data (sensor attributes and derived attributes), time-constrained data, time-constrained methods (periodic methods, sporadic methods and aperiodic methods), and a concurrency control protocol (local controller). The framework we have designed and implemented to support this model helps designers to produce real-time applications, with temporal semantics of data and transactions. It is composed of an UML profile for real-time databases, named UML-RTDB, a translator to a object-relational model, and an UML CASE Tool. UML-RTDB, based on UML2.2.1 Profiles package, contains a set of stereotypes expressing sensor attributes, derived attributes, periodic methods, sporadic methods and real-time class. So, it allows to design class diagrams for real-time databases. The translator is based on a set of mapping rules from a real-time class diagram to a object-relational model, which allows the use of simple or complex structures. Finally, the UML CASE Tool is a support for the development of real-time databases. It is built as an extension of Fujaba, an open source standard UML CASE Tool.

In our future work, we will extend UML-RTDB with other stereotypes in order to express time-constrained associations and time-constrained multiplicities. Among them, we will study how to specify real-time constraints on the proposed stereotypes, using Object Constraint Language (OCL) [14]. Moreover, we will add tools to Fujaba in order to support dynamic and behavioural model.

References

[1] 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):304–319, 2006.

[2] A. Bestavros, K.-J. Lin, and S.Son. *Real-Time Database System: Issues and Applications*, chapter Advances in Real-Time DataBase Systems Research, pages 1–14. Kluwer Academic Publishers, 1997.

[3] S. Demathieu, F. Thomas, C. André, S. Gérard, and F. Terrier. First experiments using the UML profile for MARTE. In *Proceedings of* 11^{th} *IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC'2008)*, pages 50–57, Orlando, United State, May 5-7, 2008. IEEE Computer Society.

[4] L. C. DiPippo and L. Ma. A UML package for specifying real-time objects. *Computer Standards and Interfaces*, 22(5):307–321, 2000.

[5] B. Douglass. *Real Time UML, Third Edition : Advances in The UML for Real-Time Systems*. Pearson Education, Inc, 0-321-16076-2, 2004.

[6] S. Gerard, C. Mraidha, F. Terrier, and B. Baudry. An UML-based concept for high concurrency : the real-time object. In *ISORC, 0-7695-2124-X*, pages 64–67, 2004.

[7] N. Idoudi, C. Duvallet, R. Bouaziz, B. Sadeg, and F. Gargouri. Structural model of real-time databases. In *Proc. of the 10th Intl. Conference on Enterprise Information Systems - Databases and Informations Systems topic (ICEIS'08)*, volume 1, pages 171–178, Barcelona - Spain, 2008.

[8] N. Idoudi, C. Duvallet, R. Bouaziz, B. Sadeg, and F. Gargouri. Structural model of real-time databases: an illustration. In *Proceedings of the* 11^{th} *IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC'2008)*, pages 58–65, Orlando, United State, May 5-7, 2008. IEEE Computer Society.

[9] ITU-T. Recommendation Z.109 : languages for telecommunications applications - SDL combined with UML. International Telecommunication Union, November 1999.

[10] A. Lanusse, S. Gérard, and F. Terrier. Real-time modeling with UML: The ACCORD approach. In J. Bézivin 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.

[11] C. Lu, J. Stankovich, G. Tao, and S. Son. Feedback Control Real-Time Scheduling: Framework, Modeling and Algorithms. *Real-Time Systems*, 23(1/2):85–126, 2002.

[12] OMG. "A UML Profile for MARTE, Beta 1, OMG Adopted Specification", ptc/2007-08-04, August 2007.

[13] OMG. "UML Profile for Schedulability, Performance and Time, v1.1", formal/2005-01-02, January 2005.

[14] OMG. "Object Constraint Language (OCL), v2.0", formal/06-05-01, May 2006.

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

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

[17] K. Ramamritham and C. Pu. A Formal Characterization of Epsilon Serialisability. *IEEE Transaction Journal on Knowledge and Data Engineering*, 7(6):997–1007, December 1995.

[18] K. Ramamritham, S. Son, and L. DiPippo. Real-Time Databases and Data Services. *Real-Time Systems*, 28:179–215, 2004.

[19] B. Selic. Using the object paradigm for distributed real-time systems. In *ISORC*, pages 478–480, 1998.

[20] J. Stankovic, S. Son, and J. Hansson. Misconceptions about real-time databases. *IEEE Computer*, 32(6):29–36, 1999.

[21] V. F. Wolfe, J. J. Prichard, L. C. Dipippo, and J. Black. *Real-Time Database System: Issues and Applications*, chapter The RTSORAC Real-Time-Object-Oriented DataBase Prototype, pages 279–301. Kluwer Academic Publishers, 1997.