Structural model of real-time databases: an illustration

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

Rafik Bouaziz, Faiez Gargouri
MIRACL-ISIMS, BP 1030, 3018, Sfax, Tunisie
Raf.Bouaziz, Faiez.Gargouri@fsegs.rnu.tn

Abstract

A real-time database is a database in which both the data and the operations upon the data may have timing constraints. The design of this kind of database requires the introduction of new concepts to modelize both data structures and the dynamic behavior of the database. In this paper, we propose an UML2.O profile, entitled UML-RTDB, allowing the design of structural model for a real-time database. One of the main advantages of UML-RTDB is its capacity to take into account real-time database properties through specialized concepts in rigourous, easy and expressive manner.

1. Introduction

UML is a general purpose visual modeling language for specifying, visualizing, constructing and documenting the artifacts of software systems, especially targeting the modeling of concurrent, and distributed systems. As a general modeling language, UML has the ability to be adopted to the needs of a particular application domain through the definition of stereotypes, constraints and tagged values defined by the UML profile [15]. Thus, UML has spread in areas where initially it could not find its place as the specific area of real-time applications development. 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* [8], and *ACCORD/UML* [10]. 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) [14]. 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 permanent data, but they also require management of time-constrained data and time-constrained transactions [2].

The design of real-time databases differs from the design of conventional databases. The designers of real-time databases must consider both temporal aspects of data and timing constraints of transactions [20]. The temporal consistency of data requires that the actual state of external world and the state represented by the contents of the database must be close enough to remain within the tolerance limit of applications. In general, temporal consistency of data has two aspects: *absolute* and *relative*. The absolute temporal consistency represents the requirement of data freshness, while relative temporal consistency represents the required correlation among data used together [16]. There are typically two types of timing constraints on transactions: *absolute timing constraints* (e.g. earliest start time, latest finish time) and *periodic timing constraints* (e.g. frequency of transaction initiation) [23].

The mostly used data model for real-time databases is the relational model [16]. However, due to the nature of many real-time applications that must handle complex real-world objects with short deadlines, many researchers believe that the object-oriented model is more natural and powerful than the relational model [9]. Several research projects on real-time databases have adopted the object-oriented model for building their prototype systems [23] [21]. Our work in this paper is based on the *real-time object-oriented data model* that incorporates time-constrained data and time-constrained transactions of real-time databases with the support for complex data provided by the object-oriented model. Thus, a real-time database is a collection of objects which are used to model time-critical dynamic systems in the real world. Each object has some internal state which is protected by the object abstraction. The only way

that objects can be accessed by transactions is to invoke the methods defined by objects. Objects are encapsulated in the class abstraction which facilitates modular management of resources.

There is a need to define an UML profile supporting real-time database requirements. To the best of our knowledge, there is only one based UML proposal for real-time databases modeling [4]. In their work, the authors have defined an UML package for specifying RTSORAC 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 supports the notion of Quality of Data (*QoD*) introduced in [1] which allows a robust and controlled behavior of real-time databases during the transient overloads, based on *Feedback Control Real-Time Scheduling* [13].

The contributions of this paper are five-folds.(i) We define an object model, called *real-time object*, that contains real-time attributes and real-time methods, (ii) we define an attribute model, named *real-time attribute* that supports data time semantics and *QoD* requirements, (iii) we present a general UML profile for real-time databases, called UML-RTDB, based on UML2.0 Profiles package, (iv) we define a new kind of stereotypes based on the *Evolutionary Stereotype* concept, to express dynamic semantics of real-time attribute, and (v) we define a stereotype to express real-time object features.

The remainder of the paper is organized as follows. Section 2 describes our real-time object model as well as its real-time attributes and methods. In section 3, we describe the set of stereotypes that allows to our UML-RTDB profile to express real-time database features in a structural model. In section 4, we conclude the paper and give some perspectives to our work.

2. Real-time object model

A real-time database models an external environment that changes continuously. It is designed to be kept in shared main memory for fast and predictable access [16]. Real-time objects are the real-time database entities. They represent dynamic entities of time-critical dynamic systems in the real world. Our real-time object is an extension of the real-time object as used in the ACCORD/UML approach [22], [6]. It encapsulates time-constrained data, time-constrained methods and concurrency control mechanisms. 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.

In this section, we describe the two first components of our real-time object model. We illustrate our proposal on an air traffic control system, which consists of a large collection of data describing the aircraft, their flight plans, and environment data [12]. This includes flight information such as aircraft identification, transponder code, altitude, position and speed, origin, destination, route and clearances. In our work, each aircraft in the airspace is modeled as a real-time object.

2.1. Real-time attributes

Data objects are classified into either real-time or non real-time data. A non real-time data is a classical data found in most databases, whereas a real-time data has a validity interval beyond which it becomes useless [16].

Our real-time data model is based on the model introduced in [16] and we associate to this model the notion of *maximum data error* (MDE) introduced in [1]. Thus, a real-time data is modeled by d = (d_{value}, $d_{timestamp}$, d_{avi}, d_{mde}), where d_{value} represents the real world data value, $d_{timestamp}$ is the time at which the attribute's value was last updated, d_{avi} is the absolute validity interval and d_{mde} is the maximum amount of imprecision associated with the attribute's value.

To handle the property of our real-time data model, we define an attribute model, called *real-time attribute*, that incorporates fields to support logical constraints (d_{value}), temporal constraints ($d_{timestamp}$, d_{avi}) and QoS constraints (d_{mde}). In fact, each real-time attribute is characterized by the following five fields:

- ***N**ame* (*N*): it is the name of the attribute.
- ***C**urrent **V**alue* (*CV*): is used to store the final attribute value captured by the last update correspondent method. This field is used by the system to determine logical integrity constraints of the attribute value.
- ***T**ime**S**tamp* (*TS*): is used to store the time at which the attribute's value was last updated [16]. This characteristic is of the type *Time* which is supported by the UML2.0 standard. Access to the timestamp of an attribute is necessary for determining temporal consistency of the attribute. For example, in an aircraft object, there is an attribute for storing the altitude, called *Altitude*, to which a sensor regularly provides readings. This update is expected every twenty seconds, thus the *Altitude* attribute is considered temporally inconsistent if the update does not occur within that time frame. There are many ways to define timestamps. In our model, the timestamp is the time when the value is produced. If the value is produced by a sensor device, then timestamp is the clock time when the value is read by the sensor. If the value is produced by a transaction,

timestamp is the time when the value becomes available. This field is used by system to determine whether or not timing constraints have been violated.

- ***V**alidity **D**uration* (*VD*): is used to store the *absolute validity interval* (denoted by *avi*) of the attribute value [16]. It represents the amount of time during which the attribute value is considered valid. This element is numeric and permits to determine, in association with *TS*, the *absolute consistency* of the attribute. A piece of data is considered absolutely consistent (fresh) with respect to time as long as the age of the data value is within a given interval [16]. For instance, the *Altitude* value is considered fresh if the current time is earlier than timestamp of Altitude followed by the length of the absolute validity interval (*avi*) of Altitude; i.e.$\{currenttime < Altitude.TS + Altitude.VD\}$.

- ***M**aximum **D**ata **E**rror* (*MDE*): is used to memorize the *absolute maximum data error* tolerated on the attribute value [1]. Currently, the demand for real-time database services has increased in most applications where it is desirable to execute transactions within their deadlines. They also have to use precise and fresh data in order to reflect the continuously changing external environment. However, it seems to be difficult for the transactions to both meet their deadlines and to keep the database consistent. To support these applications, the QoD concept is introduced in [1] to indicate that data stored in the database may have some deviation from its value in the real world. Thereby, data error, denoted *DE*, represents the deviation between the current data value and the updated value. The upper bound of the error is given by the *Maximum Data Error*. For instance, the maximum error on the *Speed* value is 5 km/h. This field allows the system to handle the unpredictable workload of the database by discarding sensor transactions where $DE \leqslant Attribut.MDE$, and to enhance the freshness of data using Feedback control scheduling [1]. It has the same type than *CV* field.

We note that only the two first fields; *Name* and *Current Value*, are visible to the users. The other fields are used by real-time database system in order to maintain the temporal consistency of the real-time database.

2.2. Sensor and Derived attributes

Real-time data is divided into two types: *sensor data* and *derived data* [18]. Sensor data are the data issued from sensors. Derived data are the data calculated from sensor data.

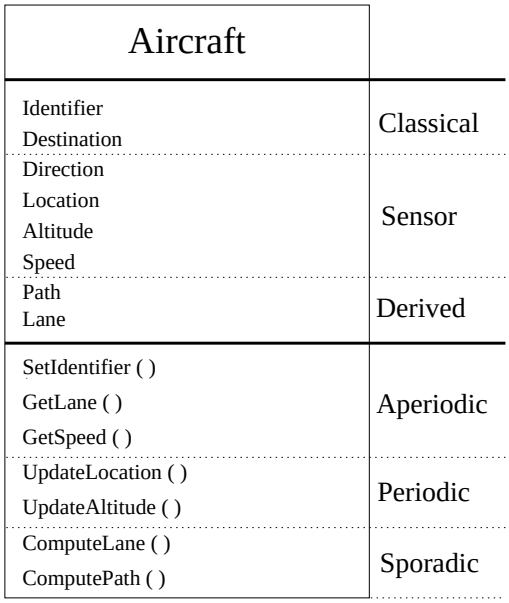

Figure 1. Aircraft object.

We characterize the real-time object model by three types of attributes as follows:

- *Classical attributes*: they are used to store a non real-time data. As shown in figure 1, we characterize the *Aircraft* object by two classical attributes, namely $Identifier$ and $Destination$.

- *Sensor attributes*: they are used to store a sensor data which must be periodically updated in order to closely reflect the real world state of the application environment.

  For example, we characterize the *Aircraft* object by four sensor attributes, namely: $Altitude$, $Speed$, $Direction$, and $Location$. These attributes are periodically updated to reflect the state of an *Aircraft* instance.

- *Derived attributes*: they are used to store a *derived data* that has to be calculated from sensor attributes. We characterize the *Aircraft* object by two derived attributes: $Lane$, which is calculated from $Location$ and $Altitude$ values, and $Path$, which is calculated from $Location$ and $Direction$ values

2.3. Real-time methods

The only way that objects can be accessed by transactions is to invoke the methods defined by objects. In our case, each method execution is considered as a transaction which may be composed of one or many sub-transactions. We classify the real-time object methods into three classes:

periodic methods, *sporadic methods*, and *aperiodic methods*.

- *Periodic methods*: the temporal consistency of each sensor data is insured by a sensor transaction, which periodically updates the value of the sensor data [18]. Thereby, we associate to each sensor attribute a method, called *periodic* method, which periodically updates the values of the $CurrentValue$ and the $TimeStamp$ fields. In our work, we assume that a periodic method execution is a sensor transaction. Sensor transactions are write-only transactions that obtain the state of the environment and write the sensed data to the database. We consider two types of timing constraints for the periodic method: *absolute timing constraints*; i.e. deadline; and *periodic timing constraint*; i.e. period. The periodicity of the method execution is dictated by the valid time of the sensor attribute value that it writes. The completion time of a periodic method must be achieved before the deadline, otherwise the value to be written will be considered obsolete [16]. Generally, the deadline of a sensor transaction is the end of each period.

  We characterize the *Aircraft* object by a set of periodic methods such as:

  - *UpdateAltitude()* which periodically carries out write operations of $CurrentValue$ and $TimeStamp$ fields of the *Altitude* attribute.
  - *UpdateLocation()* which periodically performs write operations of $CurrentValue$ and $TimeStamp$ fields of the *Location* attribute.

- *Sporadic methods*: derived data are the data calculated from sensor data [18]. Thereby, we associate to each derived attribute a method, called *sporadic* method, which sporadically calculates its value from sensor attributes. The access mode of the sporadic method to derive attribute value is always *"write"*. Its timing constraints are also *deadline* and *periodicity*. The periodicity of the method execution depends on the considered update policy. In this paper, we consider a dynamic update policy as proposed in [7]. We characterize the *Aircraft* object by the following sporadic methods:

  - *ComputeLane()* which computes the *Lane* attribute value using $Location$ and $Altitude$ values.
  - *ComputePath()* which computes the *Path* attribute value using $Location$ and $Direction$ values.

- *Aperiodic methods*: they include the remainder of methods that allow to read/write classical attributes and read only real-time attributes (sensor and derived). User transactions typically arrive aperiodically. They do not write any temporal data, but they can read/write non temporal data and only read temporal data [18]. To include nested transaction in this object model, we assume that an aperiodic method execution is a user transaction which may invoke atomic operations or invoke other methods on other objects [11]. Operations represent the actions of the method. They include statements for conditional branching, looping, I/O, and reads/writes to an attribute's *current value*, *timestamp*, *validity duration* and *maximum data error* fields.

3. An UML-RTDB profile

In this section, we present an UML profile, entitled UML-RTDB, which is a specialized variant of the UML2.0 in real-time database applications. This profile contains specialized versions of the metamodel elements; i.e. *Stereotypes*, defined in the UML2.0 metamodel. These stereotypes extend UML2.0 metamodel classes with specific attributes (sensor and derived) and a specific class (real-time object) that allow the design of class diagrams for real-time databases.

3.1. Real-Time Attribute stereotype

The main aim of the UML-RTDB is to supply to the designers of real-time databases UML extensions (stereotypes) to support both sensor and derived attributes features. A stereotype defines 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]. However, the standard stereotype can not express features of sensor and derived attributes because it does not allow the definition of structural and behavioral features [3]. Besides, the fields *timestamp*, *validity duration* and *maximum data error* represent structural features and should be expressed as ***Attributes***. Each atribute is also characterized by an update operation that ensures its freshness and must be modeled as an ***Operation***. For this reason, we base our work on the *Evolutionary Stereotype* extension mechanism of UML [3]. This extension mechanism allows the definition of new stereotypes with *structural* and *behavioral* characteristics. As shown in figure 2, the *Evolutionary Stereotype* is a metaclass which specializes the *Stereotype* metaclass of the UML 1.5 extension mechanism package. It may have structural features by defining new attributes and behavioral features by defining new operations and methods.

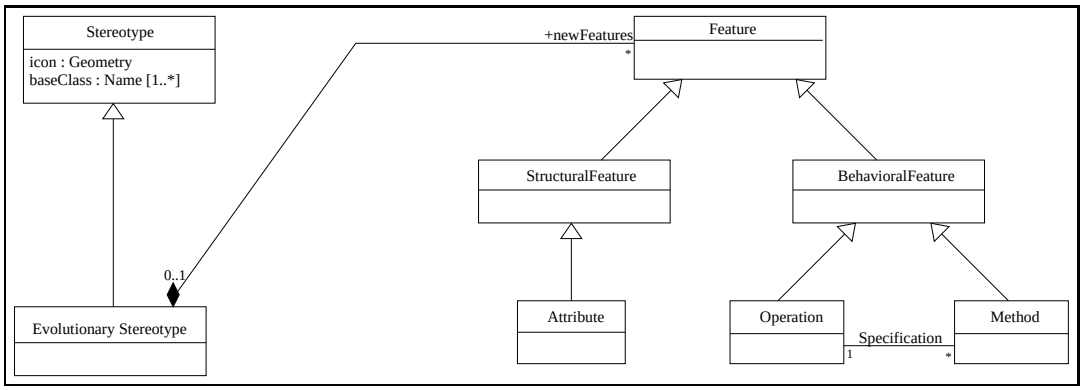

Figure 2. Abstract syntax of the $Evolutionary$ Stereotype.

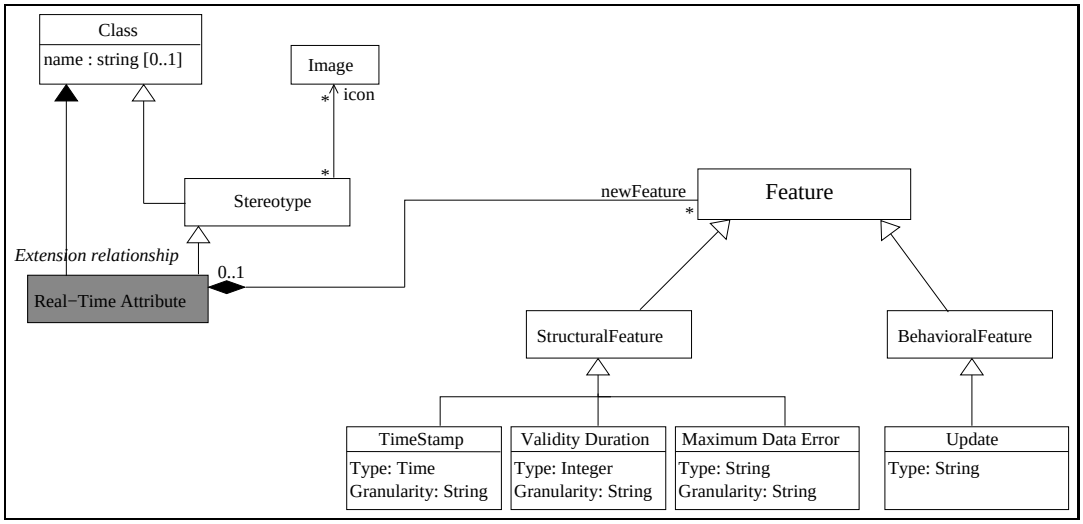

Figure 3. Abstract syntax of the ≪*RealTimeAttribute*≫ stereotype.

Moreover, we base our proposal on the Extension relationship proposed in UML2.0 Profiles package [15]. The *extension* is used to indicate that the properties of a metaclass are extended through a stereotype, and gives the ability to flexibly add (and later remove) stereotypes to (resp. from) classes. Thus, the *Stereotype::baseClass* attribute is replaced by an extension relationship to the *Class* metaclass. This former generalizes the *Stereotype* metaclass and contains a *name* attribute. In addition, the *Stereotype::icon* attribute is replaced by the *icon* role linking the *Stereotype* metaclass to *Image* metaclass.

Since the sensor attributes and derived attributes have the same structural characteristics (*Timestamp*, *Validity Duration*, *Maximum Data Error*) and behavioral characteristics (*Update operation*), we propose an abstract stereotype, called ≪*RealTimeAttribute*≫, in order to factorize these characteristics. So, instead of defining the structural and behavioral features for each stereotype aside, we define these features in a general manner within the ≪*RealTimeAttribute*≫ stereotype. This former is a realization of the Evolutionary stereotype that allows the statement of structural and behavioral features. Thereby, as shown in figure 3, the ≪*RealTimeAttribute*≫ stereotype is characterized by three structural and one behavioral characteristics. The first features are defined by ***TimeStamp***, ***Validity Duration***, and ***Maximum Data Error*** metaclasses, which specialize the *StructuralFeature* metaclass of UML metamodel. The last feature is defined by the ***Update*** metaclass, which specializes the *BehavioralFeature* metaclass.

The ***TimeStamp*** metaclass declares the timestamp of the attribute. It is characterized by two properties: *Type* and *Granularity*, where *Type* indicates the type of the Timestamp value, which is the *Time* type expressed in UML metamodel. The *Granularity* defines the granule of the Times-

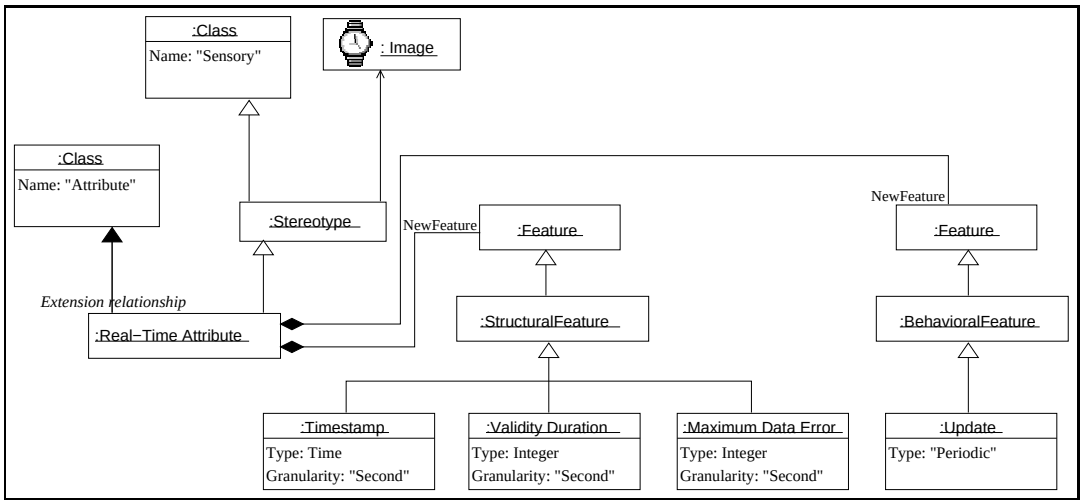

Figure 4. Abstract syntax of the ≪Sensory≫ stereotype.

tamp, which may be *"Minute"*, *"Second"*, etc.

The ***Validity Duration*** metaclass defines the valid time of the attribute value. It is characterized by two properties: *Type* and *Granularity*. The *Type* indicates the type of the Validity Duration value, which is typed *Integer*. The *Granularity* defines the granule of the Validity Duration, which may be *"Minute"*, *"Second"*, etc.

The ***Maximum Data Error*** metaclass defines the amount of error tolerated on an attribute value. It contains a *Type* property, which indicates the type of Maximum Data Error value. It has the same type as attribute value. A *Granularity* property defines the granule of the Maximum Data Error, which may be *"Minute"*, *"Second"*, etc.

The ***Update*** metaclass declares the operation which updates the $CurrentValue$ and $TimeStamp$ fields of the attribute. It contains a *Type* property that indicates the type of the operation periodicity. In our case, the operation periodicity depends on the type of the attribute. It is *"Periodic"* for a sensor attribute, and it is *"Sporadic"* for a derived attribute.

3.2 Sensory and Derived stereotypes

We define ≪*Sensory*≫ stereotype and ≪*Derived*≫ stereotype to declare respectively sensor attributes and derived attributes, in the structural model. Each stereotype presents a realization of the ≪*RealTimeAttribute*≫ stereotype according to the appropriate values of its properties, and an extension of the ***Attribute*** metaclass of UML metamodel. Figure 4 shows the abstract syntax of ≪*Sensory*≫ stereotype. Its structural features are declared by the metaclasses: *TimeStamp*, *Validity Duration*, and *Maximum Data Error*. In this work, we consider that time granularity is the *"Second"*. The designers of the real-time databases can easily modify the values of stereotype properties according to the requirements of real-time applications. The only behavioral feature of the ≪*Sensory*≫ stereotype is defined by the *Update* metaclass. The ≪*Derived*≫ stereotype has the same abstract syntax as the ≪*Sensory*≫ stereotype. They have the same structural and behavioral characteristics. Besides, they extend the same meta-instance of the *Class* metaclass, i.e. *"Attribute"*, through the *Extension relationship*. However, these two stereotypes differ in their specific notations, their meta-instances of the *Class* metaclass which generalizes the *Stereotype* metaclass and specifies the *name* of the stereotype, and their operations periodicity. So, for the ≪*Sensory*≫ stereotype, the meta-instance name of the *Class* metaclass is *"Sensory"* (cf. figure 4). For the ≪*Derived*≫ stereotype, the meta-instance name is *"Derived"*. Besides, as shown in the figure 6, we have chosen a *"watch"* as an icon declaring a sensor attribute and a *"Calculator"* declaring a derived attribute. The operation periodicity of a sensor attribute is *"Periodic"* and it is *"Sporadic"* for a derived attribute.

Figure 5 shows an instance model of the sensor attribute $Speed$, according to three layers: metamodel, users model, and user objects (user data). It is characterized as follows:

≪$Name$: Speed, $CurrentValue$: 600, $TimeStamp$: 10:25:2, $ValidityDuration$: 20, $MaximumDataError$: 5≫.

3.3. Real-Time Object stereotype

Because a real-time database is by definition a database system, it has queries, schemas, transactions, commit protocols, concurrency control support, and storage management [20]. Then, the design of a real-time database has to

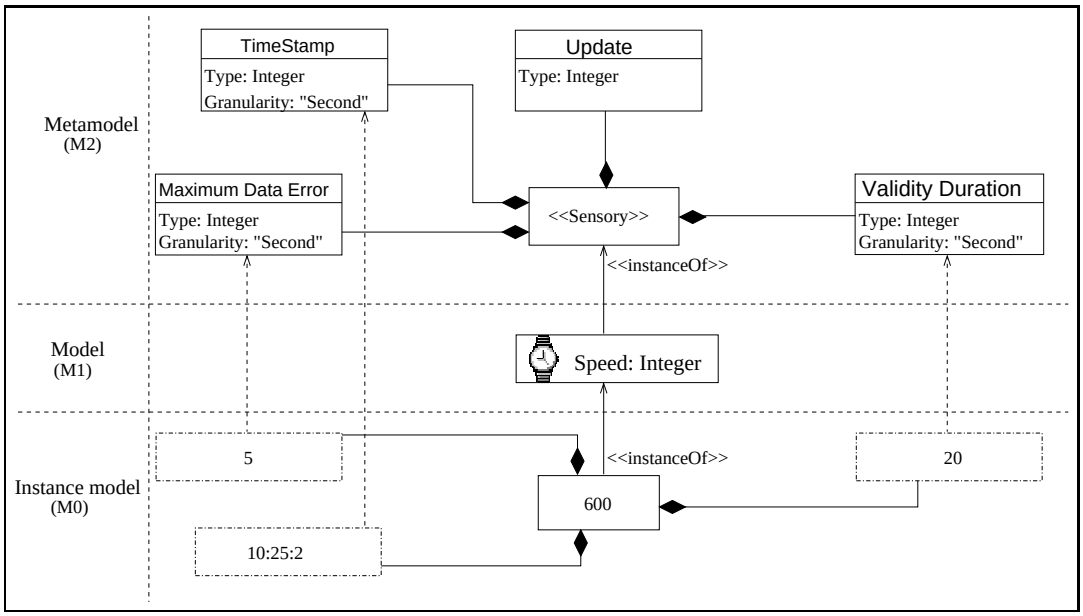

Figure 5. Instance model of the ≪Sensory≫ stereotype.

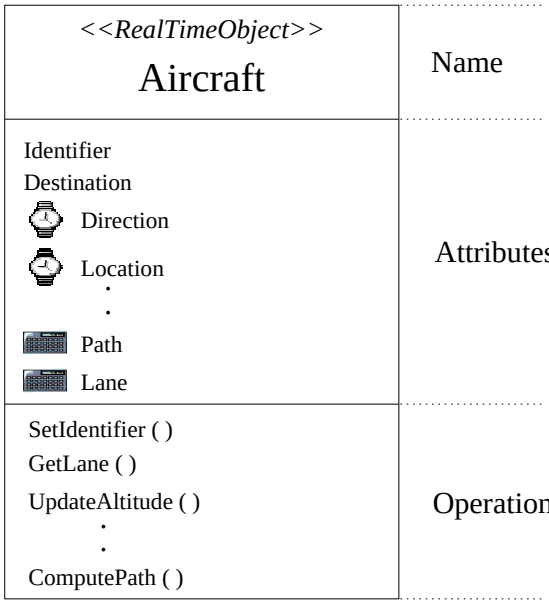

Figure 6. Aircraft real-time object class.

take into account the management of all these components. That's why, in our work, we define a ≪*RealTimeObject*≫ stereotype in order to declare the time-constrained data, the time-constrained operations, the parallelism, and the concurrency property inherent to real-time databases. The ≪*RealTimeObject*≫ 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 illustrates an *Aircraft* real-time object class. It encapsulates classical and real-time attributes (sensor and derived), and real-time operations.

4. Conclusion and future work

In this work, we have firstly presented, a real-time object-oriented data model that supports real-time database requirements. Secondly, we have proposed an UML profile for real-time databases, named UML-RTDB, based on UML2.O Profiles package, allowing the design of class models for real-time databases. The UML-RTDB contains a set of stereotypes, which are characterized by structural and behavioral features.

In our future work, we will extend the UML-RTDB with other stereotypes in order to express time-constrained associations and time-constrained multiplicities. Among then, we'll specify real-time constraints on the proposed stereotypes using **O**bject **C**onstraint **L**anguage (OCL).

5 Acknowledgement

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), 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] N. Debnath, D. Riesco, A. Maccio, G. Montejano, and P. Martellotto. Definition of new kind of UML stereotype based on OMG metamodel. In *Proceedings of the ACS/IEEE Arab International Conference on Computer Systems and Applications: AICCSA'03*, Tunis, Tunisia, 2003.

[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. A UML-based concept for high concurrency : the real-time object. In *ISORC, 0-7695-2124-X*, pages 64, 67, 2004.

[7] T. Gustafsson and J. Hansson. Dynamic on-demand updating of data in real-time database systems. In *Proceedings of ACM SAC 2004 (Symposium on Applied Computing), track on Embedded Systems: Applications, Solutions, and Techniques*, pages 846–853, 2004.

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

[9] W. Kim. *Object-Oriented Database Systems: Promises, Reality, and Future. Modern Database Systems*, pages 255–280. Addison Wesly, 1995.

[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] J. Lee, S. H. Son, and M.-J. Lee. Issues in developing object-oriented database systems for real-time applications. In *Proceeding of the IEEE Workshop on Real-Time Applications*, volume 26, pages 136–140, Washington, DC, USA, 1994. IEEE Computer Society.

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

[13] 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.

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

[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] J. Stankovic, S. Son, and J. Liebeherr. BeeHive: Global Multimedia Database Support for Dependable, Real-Time Applications. In *Real-Time Database and Information Systems*, pages 409–422. Kluwer Academic Publishers, 1997.

[22] F. Terrier, A. Lanusse, D. Bras, P. Roux, and P. Vanuxeem. Concurrent object for multitasking. In *L'objet*, volume 3, pages 179–196, Paris, France, 1997.

[23] 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.