Object-oriented Real-time Database Design based on Description Logics

Zied Ellouze[1], Nada Louati[2] and Rafik Bouaziz[2]
[1]*CES-ENIS, Sfax University, BP 1173, 3038, Sfax, Tunisia*
[2]*MIRACL-ISIMS, Sfax University, BP 1088, 3018, Sfax, Tunisia*

Keywords: Data Model, Real-time, Database, Description Logic.

Abstract: This paper proposes a Description Logics based data model of a real-time object-oriented database. It allows the modeling of structural and behavioral aspects related to the objects in a real-time database. This model provides designers a data model where they can specify both temporal aspects of data and timing constraints on transactions as well as concurrency control mechanisms.

1 INTRODUCTION

A real-time database is a time-constrained database designed to handle not only transactions with timing constraints, but also data with timing constraints (Ramamritham, 1993). The timing constraints on data are defined as how well the content of the database models the actual state of the real world while the timing constraints on transactions are expressed in the form of deadlines which indicate a certain time in the future by which the transactions must be completed. In general, there are two forms of data timing constraints: absolute and relative. The absolute timing constraint requires that a data item's age must be within a certain interval of the current time. The relative timing constraint represents the required correlation among data used together (Ramamritham, 1993). Real-time databases require also enforcement of the temporal consistency of transactions in addition to their logical consistency. Transaction logical consistency constrains the values of results produced by transactions, while transaction temporal consistency requires that transactions be treated as real-time tasks with timing constraints.

Conventional data models are not suitable for time-critical applications, since there is no mechanism to deal with time constraints. They are designed to get a good throughput or response time. Very few of them permit users to specify timing constraints. Several works have used the relational model as a data model for real-time databases (Ramamritham, 1993). Although the relational model is adequate for many applications, several researchers believe that is not suitable for real-time applications that must handle complex real-world objects with timing constraints (Ramamritham et al., 2004). Thereby, the real-time database community has migrated towards object-oriented technology. Several research projects have proposed object-oriented data models for real-time databases (Prichard et al., 1994), (Lee et al., 1994), (Perkusich et al., 1995), (Taina and Raatikainen, 1997), (Stankovic and Son, 1998), (Idoudi et al., 2010), and (Louati et al., 2011).

The data models available to object-oriented real-time designers vary in their representational and analysis strengths. This is mainly due to the inherent ambiguity in the existing means of describing them (i.e. textual and graphical). Hence, there is a dire need to introduce formalism in order to describe them accurately and allow rigorous reasoning about them. The main problem of existing real-time object-oriented data models is their lack of completeness and analysis. This is essentially because they tend to concentrate on specifying either the structural or behavioral features of data models but not both of them. In additional, most real-time object-oriented data models are informally defined so make error-prone and difficult to analyze. Formal specification of real-time object-oriented data models can enhance the understanding of their semantics. It can be used to help designers to implement formal analysis and verification on the systems described by them. In this paper, we propose a Formal Real-Time Object-Oriented (F-RTOO) data model based on Description Logics (DL). This model provides designers a data model where they can specify both temporal aspects of data and timing constraints on transactions as well as concurrency control mechanisms. Additionally, it combines the

Ellouze Z., Louati N. and Bouaziz R..
Object-oriented Real-time Database Design based on Description Logics.
DOI: 10.5220/0004453502640271
In *Proceedings of the 15th International Conference on Enterprise Information Systems* (ICEIS-2013), pages 264-271
ISBN: 978-989-8565-59-4

formal specification of structural and behavioral aspects of real-time object-oriented data models in one specification.

The structure of the rest of this paper is as follows. Section 2 discusses the related work. Section 3 provides background information on the DL. Section 4 describes our F-RTOO data model. Section 5 presents a case study, and finally in Section 6 the conclusion is presented.

2 RELATED WORKS

In recent years, several works on real-time databases have been proposed to deal with data modeling issues. Only a few of these works address real-time object-oriented data modeling.

DiPippo and Ma (Prichard et al., 1994) describe the RTSORAC model which qualifies the basic parameters for modeling a real-time database object-oriented data model. It is composed of three components: objects, relationships, and transactions.

In (Lee et al., 1994), Lee et al. describe a simple real-time object-oriented data model with atomic objects and a class manager. Atomic objects are basic entities that ensure atomicity of transactions. Class manager is the major vehicle that lessens the complexity involved in transaction management.

The G-CPN (Perkusich et al., 1995) model allows the modeling of syntactical and semantical features related to the objects in a real-time database. The semantical features can be mapped to Petri net constructions. G-CPN model has introduced extensions in an object colored Petri net to model a real-time database.

The RODAIN (Taina and Raatikainen, 1997) model is a real-time object-oriented database architecture for intelligent networks. It supports two kind of attributes: regular and real-time. A real-time object has predefined attributes for isolation level and access type. It is referenced by real-time transactions that have an explicit deadlines.

The BeeHive (Stankovic and Son, 1998) model extends traditional object-oriented data models by incorporating semantic information regarding real-time, importance, security, fault-tolerance, and QoS requirements. The BeeHive object model has some similarity in terms of the structure of objects to the RT-SORAC object model (Prichard et al., 1994). One of the main differences is that while RTSORAC model holds only real-time and approximation requirements, BeeHive model supports a rich set of types of requirements and their trade-offs.

In (Idoudi et al., 2010), Idoudi et al. describe a real-time object-oriented data model where objects contain a set of real-time attributes, a set of real-time methods, a mailbox, and a local controller. In (Idoudi et al., 2010), the data model offers solutions to manage data and transactions characteristics and concurrency. But, it does not describe the schedulability aspect.

The RTO-RTDB (Louati et al., 2011) model is a real-time object oriented database model that encapsulates time-constrained data, time-constrained transactions, and concurrency control mechanisms. It provides a very rich concepts for modeling both structural and behavioral features of a real-time database. However, it is difficult to analyze and validate the real-time database based applications it describes because of lacking precise semantics.

Table 1 presents complete analysis of different real-time object-oriented data models presented in this paragraph. They have mostly been described using textual (Prichard et al., 1994), (Lee et al., 1994), (Taina and Raatikainen, 1997), (Stankovic and Son, 1998), and (Idoudi et al., 2010) or graphical (Louati et al., 2011) notations which are not easily understood by an inexperienced designer. This leads to complications in incorporating data model concepts effectively into the modeling of a new application. To remediate to this difficulty, the solution is using an expressive notation based on DL to specify data models. This improves the data model specification quality because DL have very strong ability of representation and deduction (Baader et al., 2003).

3 A BRIEF OVERVIEW OF THE DESCRIPTION LOGICS

DL are considered the most important unifying formalism for the many object-oriented representation languages used in areas other than Knowledge Representation. These languages are equipped with a logic-based semantics (Baader et al., 2003).

3.1 Notational Conventions

The architecture of a knowledge representation system based on DL is composed of two parts: TBox and Abox. The TBox contains intensional knowledge in the form of a terminology and is built through declarations that describe general properties of concepts. The ABox contains extensional knowledge that is specific to the individuals of the domain of discourse. TBox and ABox are defined by a description language. Elementary descriptions of description language are atomic concepts and atomic roles (Baader et al., 2003). In defining the concept name by DL, it

Table 1: Tabular comparison of different object-oriented data models against RTDBs features.

	Data			Transactions						
Data Model	Logical Consistency	Absolute Temporal Consistency	Relative Temporal Consistency	Absolute Timing Constraints	Periodic Timing Constraints	Relative Timing Constraints	Scheduling	Concurrency Control	Quality of Service	Formal Semantics
RTSORAC (Prichard et al., 1994)	×	×	×	×	×		×	×		
EOODM (Lee et al., 1994)	×	×		×			×	×		
G-CPN (Perkusich et al., 1995)	×	×		×			×		×	×
RODAIN (Taina and Raatikainen, 1997)	×			×			×		×	
BeeHive (Stankovic and Son, 1998)	×	×		×	×		×	×	×	
RTO (Idoudi et al., 2010)	×	×	×	×	×		×	×		
RTO-RTDB (Louati et al., 2011)	×	×	×	×	×		×	×		

is started with an uppercase letter and then followed by the lowercase letter. Roles name starts with a lowercase letter.

We introduce now a description language, in which concepts and roles are formed according to the following syntax rule:

$C, D \rightarrow A|$; *Atomic concept*
$\top|$; *Universal concept*
$\bot|$; *Bottom concept*
$\neg A|$; *Atomic negation*
$C \sqcap D|$; *Intersection*
$C \sqcup D|$; *Union*
$\forall R.C|$; *Value restriction*
$\exists R.T|$; *Limited exitential quantification*
$R_1 \sqsubseteq R_2|$; *Role value map*

3.2 Element of Object-oriented Data Model Formalization

In this section, we first define our model of an object-oriented database. Then we formalize it using the description language proposed in (Efrizoni et al., 2010).

An object-oriented data model is a collection of classes and instances of these classes. Both instances and classes are referred to as objects. A class defines a set of attributes and a set of operations. In this model, the inheritance of properties (i.e. attributes and operations) is allowed. A relationship between two or more classes is represented by an association. This latter may be a binary association, a n-ary association, an aggregation or a composition. In the following $C(x)$: denotes the name x of a class; $a(x)$ denotes the name x of an attribute, $T(y)$ denote the type of an attribute, $P(x)$ denotes the name x of an operation of a class, and r_x denotes the name x of a role of an association.

- Class. A class is a main concept of object-oriented data model which is used to store and manage information.

$$\forall x,y.C(x) \sqsubseteq \forall a(x).T(y) \sqcap P(x)$$

- Attribute. It represents the structural feature of a class. It describes the state of an instance of an object.

$$\forall x,y.C(x) \sqsubseteq \forall a(x).T(y)$$

- Operation. It represents the behavioral feature of a class.

$$\forall x,y.C(x) \sqsubseteq \forall P_{f(x).y}.R_i(y) \sqcap (i \leqslant P_{f(x).y} \leqslant j) \quad for\ i,j \in 1 \cdots n$$

- Association. It expresses the way classes collaborate.

$$A \sqsubseteq \exists r_1.C_1 \sqcap \exists r_2.C_2$$
$$\top \sqsubseteq \forall A.C_2 \sqcap \forall A^{-}.C_1$$
$$x_1.C_1(x_1) \sqsubseteq (a \leqslant r_1^{-}.A \leqslant b)$$
$$x_2.C_2(x_2) \sqsubseteq (c \leqslant r_2^{-}.A \leqslant d)$$

- Aggregation. It can occur when a class is a collection or container of other classes.

$$\forall x_1.C_1(x_1) \sqsubseteq (\geqslant 1A)$$
$$\forall x_2.C_2(x_2) \sqsubseteq (1 \leqslant A^{-} \leqslant 1)$$

- Composition. It represents an instance of a class that become a part of instance of another class. It indicates that sometimes an object is made up of other objects.

$$\forall x.C_2(x) \in C(x) \sqcup \forall x.\neg C(x) \in C_2(x)$$

- Generalization. It depicts that one class is identified as the super class and the others as subclasses of it. Every instance of each subclass is also an instance of the super class.

$$\forall x.C_i(x) \sqsubseteq C(x) \, for \, i \in 1 \cdots n$$
$$C \sqsubseteq C_1 \sqcup C_2 \sqcup \cdots C_n$$

4 A FORMAL REAL-TIME OBJECT-ORIENTED DATA MODEL

A real-time database manipulates real-time data and executes real-time transactions (Ramamritham, 1993). Real-time data are divided into two types: sensor data and derived data. Sensor data are periodically collected from the physical environment through sensors, whereas derived data, they are computed from sensor data. When a sensor data item changes, all derived data items that are based on it need to be recomputed. Real-time transactions are classified into two categories: update transactions which are used to update values of real-time data in order to reflect the state of the real world and user transactions which represents user requests (Ramamritham et al., 2004). Update transactions are executed periodically to update sensor data, or sporadically to update derived data. User transactions arrive aperiodically. Given the added dimension of time on data and transactions, two of the interesting areas of study in real-time databases are that of transactions scheduling and concurrency control policies. Not only the schedule must meet timing constraints of transactions, it must also maintain data temporal consistency.

Hence, a real-time data model should provides support for specifying timing constraints on data and transactions, semantics of real-time data and real-time transactions, and scheduling and concurrency control mechanisms. The next subsections, proposes a formal real-time object-oriented data model, called F-RTOO, that supports these concepts.

The F-RTOO data model includes features that support the requirements of a real-time database into an extended object-oriented data model. It has a main component that models the properties of a real-time object-oriented database which is real-time object.

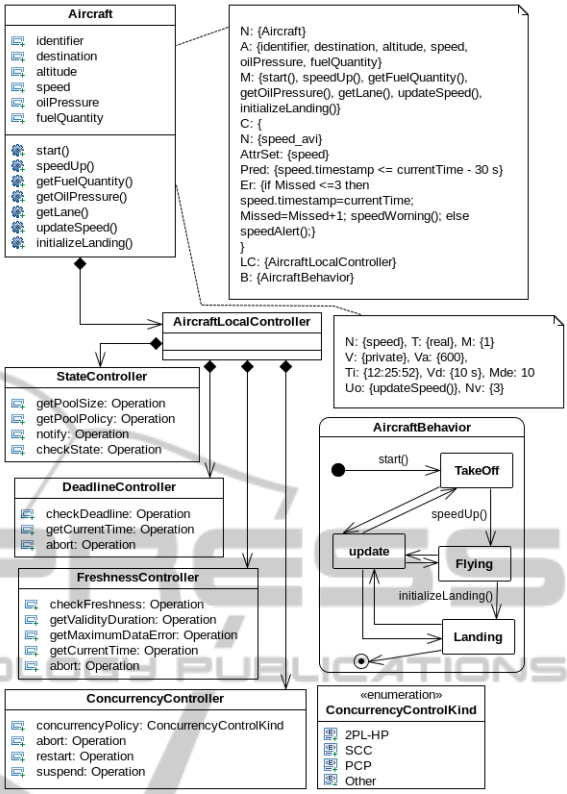

Figure 1: Aircraft real-time object.

Real-time objects represent real-time database entities. Figure 1 illustrates an example of an Aircraft real-time object for storing information about an air traffic control system in a database.

4.1 Real-time Attribute

Data objects are classified into either non real-time or real-time data (Ramamritham et al., 2004). A non real-time data is a classical data that does not become outdated due to the passage of time, whereas a real-time data has a validity interval beyond which it become useless.

The F-RTOO data model defines two types of attributes: classical attributes and real-time attributes. A classical attribute is used to store a non real-time data, while a real-time attribute stores a real-time data. In DL assertions a real-time attribute is stated as follow:

$$\forall x,y,RTA(x) \subseteq \forall n(x).string(y)$$
$$\forall x,y,RTA(x) \subseteq \forall t(x).string(y)$$
$$\forall x,y,RTA(x) \subseteq \forall m(x).integer(y)$$
$$\forall x,y,RTA(x) \subseteq \forall v(x).string(y)$$
$$\forall x,y,RTA(x) \subseteq \forall va(x).real(y)$$

$$\forall x,y,RTA(x) \subseteq \forall ti(x).dateTime(y)$$
$$\forall x,y,RTA(x) \subseteq \forall va(x).real(y)$$
$$\forall x,y,RTA(x) \subseteq \forall ti(x).dateTime(y)$$
$$\forall x,y,RTA(x) \subseteq \forall vd(x).duration(y)$$
$$\forall x,y,RTA(x) \subseteq \forall mde(x).real(y)$$
$$\forall x,y,RTA(x) \subseteq \forall uo(x).RTO(y)$$
$$\forall x,y,RTA(x) \subseteq \forall nv(x).integer(y)$$

- N (**N**ame): is the name of the attribute.
- T (**T**ype): is the type of the attribute which can be integer, real, string, etc.
- M (**M**ultiplicity): indicates how kinds of values or objects a real-time attribute can obtain.
- V (**V**isibility): represents the visibility of the attribute: public, protected or private.
- Va (**Va**lue): is used to store the real world attribute value captured by the last update correspondent method. This field is used by the system to determine the logical consistency constraints of the attribute value.
- Ti (**Ti**mestamp): is used to store the instant at which the attribute value was last updated. The timestamp determines the temporal consistency constraint of the attribute value. For example, in the Aircraft real-time object, there is a property for storing the speed, called *speed*, to which a sensor periodically provides readings. This update is expected every 30 seconds, thus the *speed* property is considered temporally inconsistent if the update does not occur within that time. The timestamp value of the *speed* property must be utilized by the real-time database system to determine that the update operation did not happen as expected. There are many ways to define timestamps (Idoudi et al., 2010). In our work, we consider that the timestamp is the time when the value is written.
- Vd (**V**alidity **d**uration): it indicates the amount of time during which the attribute value is considered valid. This field permits to determine, in association with the timestamp, the absolute consistency constraint of the real-time attribute. A real-time data is considered absolutely fresh with respect to time as long as the age of the data value is within a given interval (Ramamritham et al., 2004). For instance, the *speed* value is considered valid if the current time is earlier than the timestamp of *speed* followed by the length of the absolute validity interval of *speed*, i.e. $\{speed.Ti + speed.Vd > currentTime\}$.
- Mde (**M**aximum **d**ata **e**rror): is used to memorize the absolute maximum data error tolerated on the attribute value (Idoudi et al., 2010). This value is the upper bound of the deviation between the current attribute value in the real-time database and the reported one. Recently, the demand for real-time services has increased in most real-time database based applications where it is desirable to execute transactions within their deadlines. They also have to use fresh data in order to reflect the real world state. However, it seems to be difficult for the transactions to both meet their real-time constraints and to keep the database consistent. To support this kind of applications, the data error concept is introduced in (Amirijoo et al., 2006) to indicate that data stored in the real-time database may have some deviation from its value in the physical world.
- Uo (**U**pdate **o**peration): is used to update the value and timestamp fields of a real-time attribute. For example, in the Aircraft real-time object, there is a real-time method, called *updateSpeed()*, which periodically updates the *speed* real-time attribute.
- Nv (**N**umber of **v**ersions): is used to preserve real-time attribute version history. The multi-version attributes permits to maintain for every attribute multiple versions for a data item. This minimizes data access conflicts between real-time transactions and reduces the deadline miss ratio. In order to respect the real-time database size, the number of versions of each real-time attribute is limited. It does not have to exceed a maximum data versions number.

Note that the fields $N,T,M,V,and\ Va$ characterize classical attributes as well as real-time attributes, whereas the fields $Ti,Vd,Mde,Uo,and\ Nv$ characterize only real-time attributes. Here an example of two attributes in the Aircraft real-time object: the first is a classical attribute and the second is a real-time attribute.

$\{N = destination, T = string, M = 1, V = private, Va = Paris\}$

$\{N = speed, T = real, M = 1, V = private, Va = 600, Ti = 01/05/2012\ 10:05:23, Vd = 30s, Mde = 10, Uo = updateSpeed(), Nv = 5\}$

Real-time data are classified into either sensor or derived data (Ramamritham et al., 2004). Thereby, F-RTOO data model defines two kinds of real-time attributes: sensor attribute and derived attribute. A sensor attribute is used to store a sensor data which is periodically updated in order to reflect the real world state of the environment. A derived attribute is used to store a derived data which is sporadically updated

when a sensor attribute value, used in its computation, changes. In DL assertions a sensor and derived attribute are stated as follow:

$$\forall x.RTA(x) \subseteq Sensor(x) \sqcup Derived(x)$$
$$\forall x.Sensor(x) \subseteq RTA(x)$$
$$\forall x.Derived(x) \subseteq RTA(x)$$

4.2 Real-time Operation

The only way that objects can be accessed by transactions is to invoke the operations defined by objects. In F-RTOO data model, each operation execution is considered as a real-time transaction. A real-time operation can be stated by means of DL such:

$$\forall x,y,RTO(x) \subseteq \forall n(x).string(y)$$
$$\forall x,y,RTO(x) \subseteq \forall v(x).string(y)$$
$$\forall x,y,RTO(x) \subseteq \forall arg(x).string(y)$$
$$\forall x,y,RTO(x) \subseteq \forall exc(x).string(y)$$
$$\forall x,y,RTO(x) \subseteq \forall op(x).string(y)$$
$$\forall x,y,RTO(x) \subseteq \forall mc(x).string(y)$$
$$\forall x,y,RTO(x) \subseteq \forall mst(x).dateTime(y)$$
$$\forall x,y,RTO(x) \subseteq \forall mrd(x).duration(y)$$
$$\forall x,y,RTO(x) \subseteq \forall oct(x).dateTime(y)$$
$$\forall x,y,RTO(x) \subseteq \forall rt(x).string(y)$$
$$\forall x,y,RTO(x) \subseteq \forall per(x).duration(y)$$
$$\forall x,y,RTO(x) \subseteq \forall pri(x).integer(y)$$
$$\forall x,y,RTO(x) \subseteq \forall cp(x).concurrencyKind(y)$$

- N (**N**ame): denotes the name of the real-time method.
- V (**V**isibility): indicates whether a real-time method is public, protected, or private.
- Arg (**Arg**uments): is a set of arguments for the real-time method, where each argument has the same structure as an attribute, and is used to pass information in the method.
- Exc (**Exc**eptions): is a set of exceptions that may be raised by the real-time method to signal that the method has terminated abnormally.
- Op (**Op**erations): is a set of operations which represent the impementation of the method.
- Mc (**M**ethod **c**onstraints): is a set of methods constraints. A method constraint is of the form $< N, OpSet, Pred, Er >$ where N is the name of the method constraint, OpSet represents a subset of the operations in Op, Pred is a boolean expression which is specified over OpSet to express execution constraints, timing constraint and precedence constraints, and Er is a enforcement rule. The enforcement rules are used to specify the actions to take if the predicate (i.e. Pred) evaluates to false. A complete definition of an enforcement rule is described in the next subsection on constraints. Here is an example of a method constraint predicate in the Aircraft real-time object:

  $$Pred : getLane().Mct < currentTime + 5s$$

  A deadline of currentTime +5s has been specified for the completion of the *getLane()* method. Note the use of the *Mct* property which represents the completion time of the executable method.
- Mst (**M**ethod **s**tart **t**ime): indicates the execution start time of the real-time method.
- Mrd (**M**ethod **r**elative **d**eadline): specifies the deadline of a method execution.
- Mct (**M**ethod **c**ompletion **t**ime): indicates the time at which the method finishes its execution.
- Rt (**R**eturn **t**ype): specifies which kinds of value or object a real-time method can return.
- Per (**Per**iod): indicates the frequency of the real-time method initiation.
- Pri (**Pri**ority): Priority specifies the priority order of a real-time method.
- Cp (**C**oncurrency **p**olicy): specifies the concurrency policy of a real-time method. A concurrency policy may be reader, writer or parallel (Louati et al., 2011). A reader real-time method implies that multiple calls from concurrent methods may occur simultaneously and will be executed simultaneously if there is no writer methods using one or more data that the reader method needs. A writer real-time method implies that multiple calls from concurrent methods may occur simultaneously and will be treated as soon as concurrency on data permits its execution. A parallel real-time method is a method whose actions do not use any data of the database in reading mode nor in writing mode.

Real-time transactions are classified into three categories: periodic transactions, aperiodic transactions and sporadic transactions (Ramamritham et al., 2004). Thereby, F-RTOO data model specifies three types of operations: periodic operations, aperiodic operations, and sporadic operations. Periodic operations update periodically sensor data. They are write-only operations that obtain the state of the real world and write the sensed data to the database. Sporadic operations calculate sporadically derived data. The access mode of the sporadic operation to derived data is always write. Aperiodic operations do not write any real-time data, but they can read/write non real-time

data and only read real-time data. In DL assertions periodic, aperiodic and sporadic real-time operations are stated as follow:

$$\forall x.RTO(x) \subseteq Periodic(x) \sqcup Aperiodic(x) \sqcup Sporadic(x)$$
$$\forall x.Periodic(x) \subseteq RTO(x)$$
$$\forall x.Aperiodic(x) \subseteq RTO(x)$$
$$\forall x.Sporadic(x) \subseteq RTO(x)$$

4.3 Real-time Class

The design of a real-time database, which is by definition a database system, has to take into account the management of many components such as queries, schemas, transactions, commit protocols, concurrency control protocol, and storage management (Stankovic et al., 1999). In order to deal with time-constrained data, time-constrained operations, parallelism, and concurrency property inherent to real-time databases, we introduce the real-time class concept. This latter specifies that instances of a class will encapsulate real-time attributes and real-time operations and a local concurrency mechanism. Because of the dynamic nature of the real world, more than one operations may send requests to the same real-time class. Concurrent execution of these operations allows several methods to run concurrently within the same class. To handle this essential property of real-time database systems, we associate to each real-time class a local concurrency control mechanism, that manages the concurrent execution of its operations. Thus, the class receives messages (or requests) awaking its local controller that checks the timing constraint attached to messages and selects one message following a special scheduling algorithm. The local controller verifies the concurrency constraints with the already running methods of the object. Then, it allocates a new thread to handle the message when possible. When an operation terminates its execution, the corresponding thread is released and concurrency constraints are relaxed (Louati et al., 2011).

The following DL assertions state that a real-time class is composed of: a set of real-time attributes, a set of real-time operations, and a local controller.

$$\forall x.LC(x) \in RTC(x) \sqcup \forall x.\neg RTC(x) \in LC(x)$$
$$\forall x.RTA(x) \in RTC(x) \sqcup \forall x.\neg RTC(x) \in RTA(x)$$
$$\forall x.RTO(x) \in RTC(x) \sqcup \forall x.\neg RTC(x) \in RTO(x)$$

5 A CASE STUDY

Throughout, this section, we will use a running example to prove relevance of our propositions. We illustrate our proposal on a freeway traffic control system. As depicted in the Figure 2, the freeway traffic control system architecture consists of three entities respectively dedicated to: represent a transport infrastructure road that links two conurbations (*RoadLink*), depict a part of a route (*RoadSegment*), and represent physical entity (*Vehicle*). *RoadSegment* and *Vehicle* represent the description of two physical elements that are supervised by the controller. In addition, they are characterized by one or more real-time data which could determine theirs evolution. These real-time data are classified into either sensor data or derived data. In fact, each vehicle has two sensor data (i.e. *speed* and *length*) which are periodically updated to reflect its state and each road segment has two derived data (i.e. *trafficVolume*, and *trafficOccupancy*) which are calculated from sensor data.

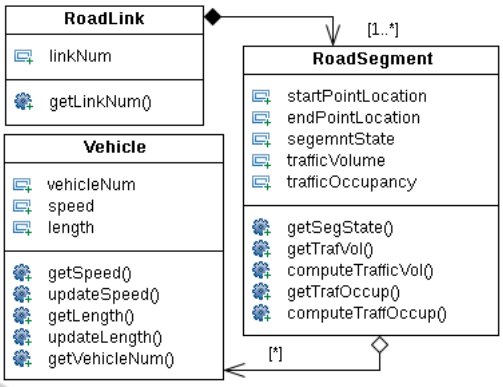

Figure 2: Freeway Traffic Control System Class Diagram.

Definition of properties of RoadLink's class capture attributes and operations as follows:

$$\forall x,y.RoadLink(x) \subseteq \forall linkNum(x).integer)$$
$$\forall x,y.RoadLink(x) \subseteq \forall P_{getLinkNum()}.integer(y) \sqcap (\leqslant 1\ P_{getLinkNum()})$$

The same manner defines attributes and operations for each other classes (i.e. Vehicle and RoadSegment).

$$\forall x,y.Vehicle(x) \subseteq \forall vehicleNum(x).integer(y)$$
$$\forall x,y.Vehicle(x) \subseteq \forall speed(x).real(y)$$
$$\forall x,y.Vehicle(x) \subseteq \forall length(x).real(y)$$
$$\forall x,y.Vehicle(x) \subseteq \forall P_{getSpeed()}.real(y) \sqcap (\leqslant 1\ P_{getSpeed()})$$
$$\forall x.Vehicle(x) \subseteq \forall P_{updateSpeed()} \sqcap (\leqslant 1\ P_{updateSpeed()})$$
$$\forall x,y.Vehicle(x) \subseteq \forall P_{getLength()}.real(y) \sqcap (\leqslant 1\ P_{getLength()})$$

$$\forall x.Vehicle(x) \subseteq \forall P_{updateLength()} \sqcap (\leqslant 1\ P_{updateLength()})$$
$$\forall x,y.Vehicle(x) \subseteq \forall P_{getVehicleNum()}.integer(y) \sqcap (\leqslant 1\ P_{getVehicleNum()})r$$
$$\forall x,y.RoadSegment(x) \subseteq \forall startPointLocation(x).integer(y)$$
$$\forall x,y.RoadSegment(x) \subseteq \forall endPointLocation(x).integer(y)$$
$$\forall x,y.RoadSegment(x) \subseteq \forall segmentState(x).string(y)$$
$$\forall x,y.RoadSegment(x) \subseteq \forall traficVolume(x).real(y)$$
$$\forall x,y.RoadSegment(x) \subseteq \forall traficOccupancy(x).real(y)$$
$$\forall x,y.RoadSegment(x) \subseteq \forall P_{getSegState()}.string(y) \sqcap (\leqslant 1\ P_{getSegState()})$$
$$\forall x.RoadSegment(x) \subseteq \forall P_{computeTrafVol()} \sqcap (\leqslant 1\ P_{computeTrafVol()})$$
$$\forall x,y.RoadSegment(x) \subseteq \forall P_{getTrafVol()}.string(y) \sqcap (\leqslant 1\ P_{getTrafVol()})$$
$$\forall x.RoadSegment(x) \subseteq \forall P_{computeTrafOccu()} \sqcap (\leqslant 1\ P_{computeTrafOcc()})$$
$$\forall x,y.RoadSegment(x) \subseteq \forall P_{getTrafOccu()}.string(y) \sqcap (\leqslant 1\ P_{getTrafOccu()})$$

6 CONCLUSIONS

This paper has shown the formalization of a real-time object-oriented data model in term of specified logic i.e Description Logics. Through this formalization, the deductive capabilities of DL have been exploited. Moreover, the proposed data model not only capture the structural aspects of a real-time database features, but also the behavioral aspects. Additionally, it provides designers a data model where they can specify both temporal aspects of data and timing constraints on transactions as well as concurrency control mechanism.

We are currently working on the implementation of a real-time query language that supports our real-time object-oriented data model.

REFERENCES

Amirijoo, M., Hansson, J., and Son, S. H. (2006). Specification and management of qos in real-time databases supporting imprecise computations. *IEEE Trans. Computers*, 55(3):304–319.

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

Efrizoni, L., Wan-Kadir, W., and Mohamad, R. (2010). Formalization of uml class diagram using description logics. In *Information Technology (ITSim), 2010 International Symposium in*, volume 3, pages 1168–1173.

Idoudi, N., Louati, N., Duvallet, C., Sadeg, B., Bouaziz, R., and Gargouri, F. (2010). A framework to model real-time databases. *International Journal of Computing and Information Sciences (IJCIS)*, 7(1):1–11.

Lee, J., Son, S. H., and Lee, M.-J. (1994). 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. IEEE Computer Society.

Louati, N., Duvallet, C., Bouaziz, R., and Sadeg, B. (2011). RTO-RTDB: A real-time object-oriented database model. In *In Proceedings of the International Conference on Parallel and Distributed Computing and Systems*. ACTA Press.

Perkusich, M. L., de Fatima, M., Turnell, Q., and Perkusich, A. (1995). Object-oriented real-time database design based on petri nets. In *In Proceedings of the International Workshop on Active and Real-Time Database Systems*, pages 104–121. Springer.

Prichard, J., DiPippo, L., Packham, J., and Fay-Wolfe, V. (1994). RTSORAC: A Real-Time Object-Oriented Database Model. In Springer-Verlag, editor, *Proc. of the 5^{th} Intl. Conf. on Database and Expert Systems Applications (DEXA'94)*, pages 601–610, London, UK.

Ramamritham, K. (1993). Real-time databases. *Distributed and Parallel Databases*, 1(2):199–226.

Ramamritham, K., Son, S. H., and DiPippo, L. C. (2004). Real-time databases and data services. *Real-Time Systems*, 28(2-3):179–215.

Stankovic, J. A. and Son, S. H. (1998). Architecture and object model for distributed object-oriented real-time databases. In *ISORC*, pages 414–424.

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

Taina, J. and Raatikainen, K. (1997). Rodain: a real-time object-oriented database system for telecommunications. In *Proceedings of the workshop on on Databases: active and real-time*, CIKM '96, pages 10–14, New York, NY, USA. ACM.