A UML Package for Specifying Real-Time Objects*

Lisa Cingiser DiPippo
Lynn Ma
The University of Rhode Island
Kingston, RI USA 02881
lastname@cs.uri.edu

Abstract

The Unified Modeling language provides a robust set of tools for modeling software systems. However, these tools do not directly address the requirements of real-time systems. Many real-time systems require the specification of data that has strict timing constraints. This paper presents a UML package for specifying real-time objects called RT-Object. The constructs in the package are based on the objects of the RTSORAC (Real-Time Semantic Objects Relationships And Constraints) model. The RT-Object package has been used to design real-time objects in a Real-Time Multi-User Virtual Environment in which widely distributed users collaborate in time-critical planning and decision making.

1 Introduction

The Unified Modeling Language (UML) [1,2] is a general object-based metamodel that provides mechanisms for specifying a wide variety of systems. It provides a set of diagram types that enable system designers to visually model the target system. Real-time systems designers use UML to specify system models from requirements analysis through implementation. However, UML does not directly address the specific modeling concerns of real-time systems. For instance, UML does not currently provide metamodel constructs for specifying real-time objects – objects that represent dynamic entities in the environment. Such objects require mechanisms for specifying constraints on attributes, both temporal and logical. Real-time objects may also require special concurrency control specification in order to take timeliness into account when making decisions.

The Object Management Group (OMG), the standardizing body for UML, has recently put out a Request for Proposals (RPF) for a UML Profile for Scheduling, Performance, and Time [3]. The RFP is looking for proposals that specify standard paradigms of use for modeling time, performance and schedulability. Many of the required features specified in the RFP will provide a foundation for the definition of a real-time object package. For example, the RFP requires provision of facilities for modeling time. It also requires a means to model deadlines, execution times and other timing constraints that will be necessary when specifying the real-time object construct.

This paper proposes a UML package, called RT-Object, consisting of real-time object elements that can be used by real-time systems designers to specify a wide variety of real-time objects. The package elements extend UML metamodel classes with specific attributes and constraints for real-time systems. As an example, consider an air-traffic control system, in which each airplane in the airspace can be modeled as a

* This work is partially supported by the U.S. Office of Naval Research grant N000149610401.

real-time object. The designer of the system can include the UML RT-Object package when building their design. Then any real-time objects in the system, like the airplane object, can extend the classes in the package with application specific features. This alleviates from the system designers the necessity of defining real-time features, such as deadlines, and worst case execution times, that are common to most real-time objects.

The remainder of this paper is organized as follows. Section 2 provides background and related work in UML, real-time extensions to UML, and the real-time object model on which this work is based. Section 3 presents the RT-Object package, first describing the metamodel features that it extends, and then describing the new elements that we have defined. Section 4 provides an example of how the RT-Object package was used in the design of a real-time multi-user virtual environment application. Section 5 concludes by summarizing the contributions of this work, and discussing some future work.

2 Background

This section provides background on the UML modeling language and the mechanisms that are used to create the RT-Object package. It also reviews other work that has been done in extending UML for real-time system specification. Finally, the section describes the RTSORAC (Real-Time Semantic Object Relationships and Constraints) model for real-time objects on which the RT-Object package is based.

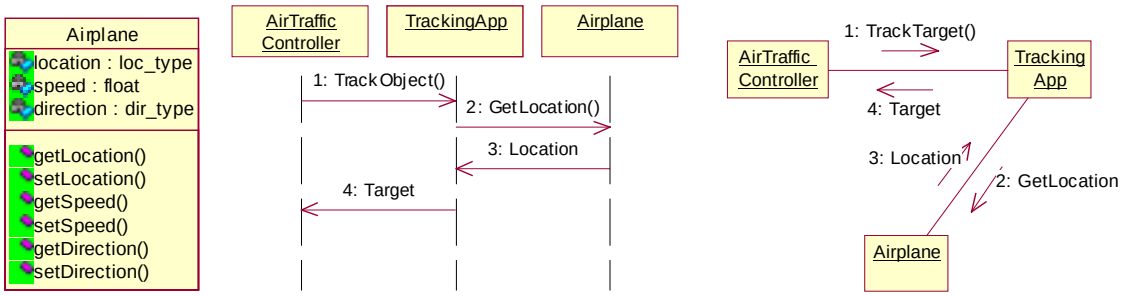
2.1 UML

UML is a third-generation object-oriented modeling language for specifying, constructing, visualizing, and documenting software systems [2]. The original designers of UML incorporated the object-oriented community's consensus on core modeling concepts, with the ability to specify extensions for specific types of applications. System designers use UML diagrams to develop software. The diagrams can be augmented with constraints, expressed in a formal constraint language, to more precisely express the semantics of the application.

The UML metamodel defines the constructs that designers can use in modeling software systems. The metamodel is represented by a Foundation Package that contains all of the constructs provided by UML for modeling software systems. The constructs of the Foundation Package are described using UML diagrams.

2.1.1 Diagrams

Several types of diagrams are provided by UML for visualizing the design of a software system. We will describe several of these diagrams that can be useful for specifying real-time systems. A *class diagram* is used to model the important abstractions in the system, and how they relate to each other. Class diagrams can model entities and relationships. A class diagram specifies the attributes and operations that objects of this class exhibit. Figure 1a illustrates a simple airplane class with attributes such as speed, location, and



(a) (b) (c)

Figure 1 - Example Airplane Diagrams

direction, and operations to update and read the attributes. An object diagram is similar to a class diagram, except that it represent an object, or an instance of a specified class. An object diagram can be used to express a snapshot of an instance of a class within a particular scenario.

A scenario is a particular path through the system functionality. UML provides two types of diagrams to illustrate scenarios: *sequence diagrams* and *collaboration diagrams*. A sequence diagram shows the sequence of messages between objects. Messages passed between objects are modeled with horizontal lines. The passage of time is indicated by vertical position in the diagram. Figure 1b shows a sequence diagram for an airplane tracking operation. In this application, a human user makes a request to track a particular target. The *TargetApp* object receives the request and calls *GetLocation* on one or more *Airplane* objects. The *Airplane* returns its location to the *TargetApp* object, which aggregates its results and sends a target back to the user.

A collaboration diagram shows a view of the interactions or structural relationships that occur between objects in the model. Figure 1c shows the same airplane tracking example in a collaboration diagram. The difference between the sequence diagram and the collaboration diagram is that while the collaboration diagram specifies the relationships among the classes in a scenario, the sequence diagram specifies the timing involved in the scenario. This can be useful when designing real-time systems.

2.1.2 Constraints

Constraints in UML are expressed using the Object Constraint Language (OCL). OCL is designed to augment class diagrams with additional information that cannot be otherwise expressed by UML diagrams. OCL allows the definition of both metamodel constraints, and user-level constraints. The main purpose of OCL is to specify restrictions on the possible system states with respect to a given model [4]. OCL constraints are declarative. That is, they express what the constraint is, but does not specify how it should be maintained.

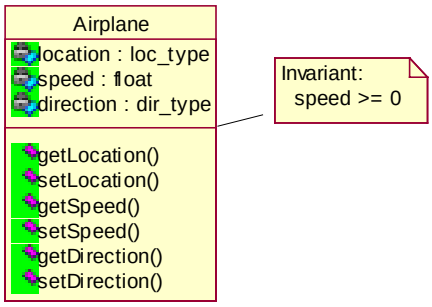


Figure 2 - Airplane Constraints

According to the UML metamodel, any kind of ModelElement (one of the the most general structures in the UML metamodel) can be associated with a constraint. Figure 2 shows the *Airplane* class diagram of Figure 1a augmented with a constraint that specifies that each airplane in the tracking application has a *speed > 0*. Notice that the constraint is a notation associated with a class, and not a part of the class itself. For a full description of the expressiveness of OCL, see [5].

2.1.3 Foundation Package

The UML Foundation Package is the infrastructure for UML. It is made up of three subpackages: the Core, the Extension Mechanisms, and the Data Types [1]. The Core package defines the basic abstract and concrete constructs needed for the development of object models. Figure 3 displays the backbone of the Core package. As the diagram illustrates, any entity in a model is derived from the general class *ModelElement*. Notice the relationship between *ModelElement* and *Constraint*. As stated in Section 2.1.2, a constraint can be expressed on any entity in the model, by associating it with that element. The Foundation package also expresses the basic constructs of object modeling, such as classifier, attribute, method, operation, etc. In an application of these metamodel constructs, the associations among them are displayed in class diagrams, such as in Figure 1a, where the attributes and operations are expressed as an integral part of the class.

The Extension Mechanisms package specifies how the model elements can be customized and extended. It defines the semantics for stereotypes, constraints and tagged values [1]. A stereotype extends an existing model element by introducing additional values, constraints, and graphical representations. It shares the attributes, associations, and operations of its base class, and cannot add others. Creating stereotypes allows users to customize model elements for a particular application. A tagged value allows arbitrary information to be attached to a model element. Tagged values are often associated with stereotypes to specify extra information required by the specific application.

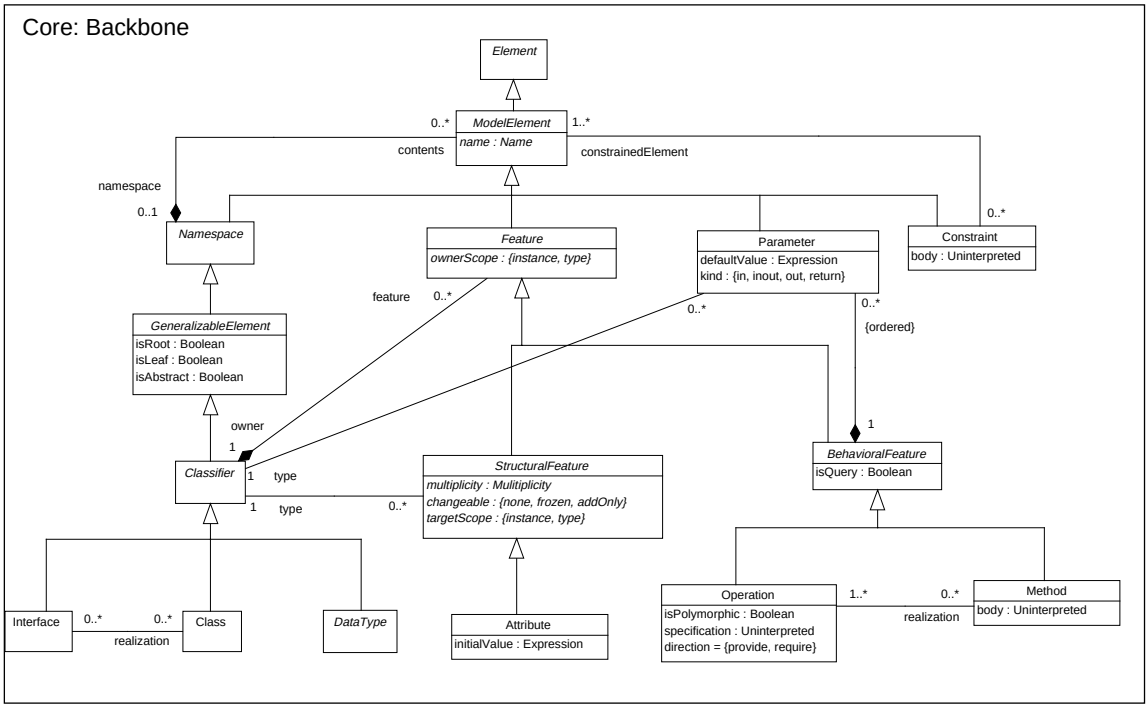


Figure 3 - UML Core Package Backbone

2.2 Real-Time UML

The OMG's standardization effort towards a real-time UML has been motivated by a lack of direct support in the standard for key elements necessary to real-time systems design. The RFP requires that responses include mechanisms to handle real-time features, such as the following [3]:

1. **Models of physical time**. Responses should include models for universal time, mission time, discrete and continuous time, global and local time.

2. **Timing Specifications.** Responses should include modeling of timing specifications such as deadlines, periods, frequencies, execution times, time budgets, and inter-arrival times should be included.

3. **Timing Facilities**. Responses should include specification for timing facilities such as time resolution, synchronization, timer object, and clocks.

4. **Resource Management**. Responses should provide a means to model physical resources, such as CPU, and non-physical resources, such as buffers. They should also specify techniques for managing resources.

5. **Concurrency**. Responses should provide a way to specify various concurrency schemes.

Several groups are working on responses to this RFP that meet these requirements. Along with the groups working on a response to the RFP, other work has been done recently towards defining requirements for real-time UML. In [6], a formal UML package is presented for specifying real-time system constraints. The package consists of UML constructs representing abstractions of a real-time clock, a timer, a process, a resource and a precedence constraint. These package elements provide a real-time system designer with basic real-time features with which to build a design. The work in [7] reviews the state of the art of a new generation of object technologies that are addressing object-oriented real-time systems modeling - most notably the UML. The paper discusses how these technologies are evolving to meet the needs of real-time developers

The work in [8] discusses highlights of UML as they apply to the design of real-time systems. This work does not attempt to extend UML, but rather finds ways in which to use the existing standard to model real-time systems. This involves expression of timing constraints on operations that are specified in UML. For instance, they show how timing annotations in the form of constraints can be placed on sequence diagrams and on collaboration diagrams to express deadlines and other timing constraints on operations. While all of this previous work is important for the development of a real-time UML, none of it addresses the issue of how to model data objects that have real-time characteristics.

2.3 Real-Time Objects in the RTSORAC Model

We have developed a model for real-time objects, called the RTSORAC model (Real-Time Semantic Objects, Relationships And Constraints) [9]. This model was originally designed for real-time objects in a real-time object-oriented database (RTOODB). However, it is general enough to be used to describe real-time objects in any real-time system. The constructs contained in the RT-Object UML package that we present in Section 3 are based on the RTSORAC model.

An object in the RTSORAC model consists of five components: *Name*, *Attributes*, *Methods*, *Constraints*, and *Compatibility Function*. The *Name* of an object represents a unique identifier. In the following sections we will describe each of the other four features.

Figure 4 displays a real-time hurricane object in a weather tracking application. As new data comes in from sensors, satellites, and other sources, the object is updated. Users of the system can access the data object, and may make changes to it with information that may not be available from the other sources.

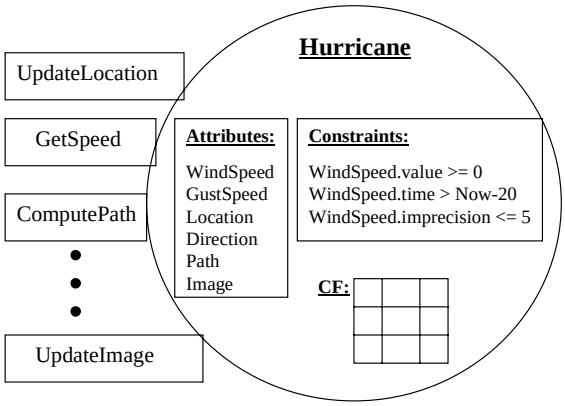


Figure 4 - Example Hurricane Object

2.3.1 Attributes

Each object in the RTSORAC model has a set of attributes that represent the properties of the data object. In the hurricane example, the attributes include WindSpeed, GustSpeed, Direction, Location, and Path. Each attribute is made up of three parts: *Value*, *Time*, and *Imprecision*. The *Value* of an attribute represents its current state through a specific data type. For instance, the *Value* of WindSpeed is an integer represented in miles per hour, or kilometers per hour.

The *Time* field of an attribute is a timestamp that represents the age of the attribute. That is, it provides a way to determine how old the *Value* of the attribute is. The *Time* field usually represents the time at which the attribute's value was last updated. This information can be used to determine if timing constraints have been violated. The *Imprecision* field of an attribute provides a placeholder that allows for algorithms and techniques that may use imprecise computation [10], or relaxed serialzability in concurrency control [11, 12, 13]. Such techniques may allow certain amounts of imprecision to be introduced into the data. By storing this imprecision information in the attribute, we can provide mechanisms for keeping the amount of imprecision bounded [11], and for updating the attribute with precise data.

2.3.2 Methods

Each object in the RTSORAC model has a set of methods that provides the means for other objects to access its state. A method has a set of arguments, a sequence of operations, a specified execution time, and a set of operation constraints. Each method argument has the same structure as an attribute, having a *Value*, a *Time* and an *Imprecision* field. This allows all of the parts of the attributes to be updated, and accessed. The worst case execution time of the method is specified to allow real-time scheduling and concurrency control algorithms to perform timing analysis. The model also expresses a set of constraints on the execution of the method including absolute timing constraints on the method as a whole, or on a subset of operations within the method. In Figure 4, the methods of the Hurricane object include

UpdateLocation, *GetSpeed*, and *ComputePath*. Notice that some methods, like *UpdateLocation*, access single attributes, while others access more than one. *GetSpeed* returns both *WindSpeed* and *GustSpeed*.

A method may additionally have associated with it a pair of sets of attributes indicating which attributes it reads (*read affected set*) and which attributes it writes (*write affected set*) [11]. For example, the Hurricane object's GetSpeed method reads both the WindSpeed attribute and the GustSpeed attribute, so its read affected set is {WindSpeed, GustSpeed}. Since this method does not write any attributes, its write affected set is empty. This information can be used to provide fine-grained concurrency control. That is, locking can be done at the method level rather than at the object level. Section 2.3.4 discusses the use of the affected sets further.

2.3.3 Constraints

Constraints in a RTSORAC object define the correct states the object. A constraint can be specified on any attribute, or set of attributes in the object. More specifically, a constraint can include reference to any of the fields of an attribute. This provides a mechanism for specifying logical integrity constraints, using the *Value* field, timing constraints, using the *Time* field, and imprecision constraints, using the *Imprecision* field. For instance, in Figure 4 the constraint *WindSpeed.value >= 0* expresses a logical integrity constraint on the *WindSpeed* attribute. The constraint *WindSpeed.time > Now-20* is a timing constraint requiring that the value of the *WindSpeed* attribute not be more than twenty seconds old. In order to bound the amount of imprecision that might be introduced into the *WindSpeed* attribute to 5 mph, the following constraint is expressed: *WindSpeed.Imprecision <= 5.*

Constraints can also be expressed on sets of attributes. This allows the specification of interattribute timing constraints. This is necessary if the value of one attribute is computed based on the values of several other attributes. In order to ensure that the derived attribute value is valid, we need to constrain the values of the attributes that derive it to be approximately the same age. We can specify a relative timing constraint. For example, in the Hurricane object, to compute the Path attribute, we use the Location and the Direction attributes. The requires that in order to ensure the validity of the value of Path, Direction and Location must be updated at close to the same time. This requirement can be expressed with the following constraints: $|Location.time - Direction.time| < 10$.

2.3.4 Compatibility Function

The compatibility function of a RTSORAC object provides a mechanism to express which methods can execute concurrently. It is represented as a *MxM* matrix where *M* represents the number of methods in the object. Each entry in the matrix indicates whether or not the corresponding methods can execute concurrently. This information can be used to provide fine-grained concurrency control that does not necessarily lock the entire object when a method is called. In real-time systems, fine-grained locking can provide increased concurrency which can allow more timing constraints to be met [11].

Several algorithms have been developed that use the affected sets of a method to determine which methods can execute concurrently based on which attributes are being read and written by the methods [11,14 ,15]. The compatibility function matrix entries can be simple Boolean values, as in [14], or they can be arbitrary Boolean expressions that use dynamic information to determine the compatibility of the corresponding methods [11]. In the latter case, the expressed compatibility may relax serializability in order to meet timing constraints. This could introduce imprecision into the attributes, and the concurrency control should provide a mechanism for bounding it.

3 A UML Package for Real-Time Objects.

In this section we present a set of constructs, in the form of a UML package, that can be used to model real-time objects. Any real-time system application can import this package and extend its features to represent real-time objects in the application. The package includes several class definitions that extend classes in the UML Core Package of the metamodel. Figure 5 displays these classes and their associated constraints.

In the following sections we first present several assumptions that we make in our model of real-time objects. These assumptions are based upon requirements specified in the RFP for a UML Profile for Scheduling, Performance, and Time [3]. We then describe the existing features of the UML metamodel that are extended in our UML package for real-time objects. We go on to describe the classes and associated constraints that represent the features of a real-time object. We show how these features follow the RTSORAC model.

3.1 Assumptions

The RFP for a UML Profile for Scheduling, Performance, and Time [3] has several mandatory requirements for features that we will assume to be available for our model. The RFP requires modeling facilities that allow users to represent various common models of time. These include universal (real-world) time, discrete and continuous time, global and localized time, time intervals, and duration. According to the UML Enhanced Views of Time RFP [16], a *Time* object is an instant relative to a particular view of time. Our model described in Section 3.3 assumes that a Time object can take on any of the above representations of time.

The scheduling RFP also requires a means to model timing specifications such as deadlines, periods, frequencies, response times, and execution times. We use these timing specifications to help form our model of real-time objects.

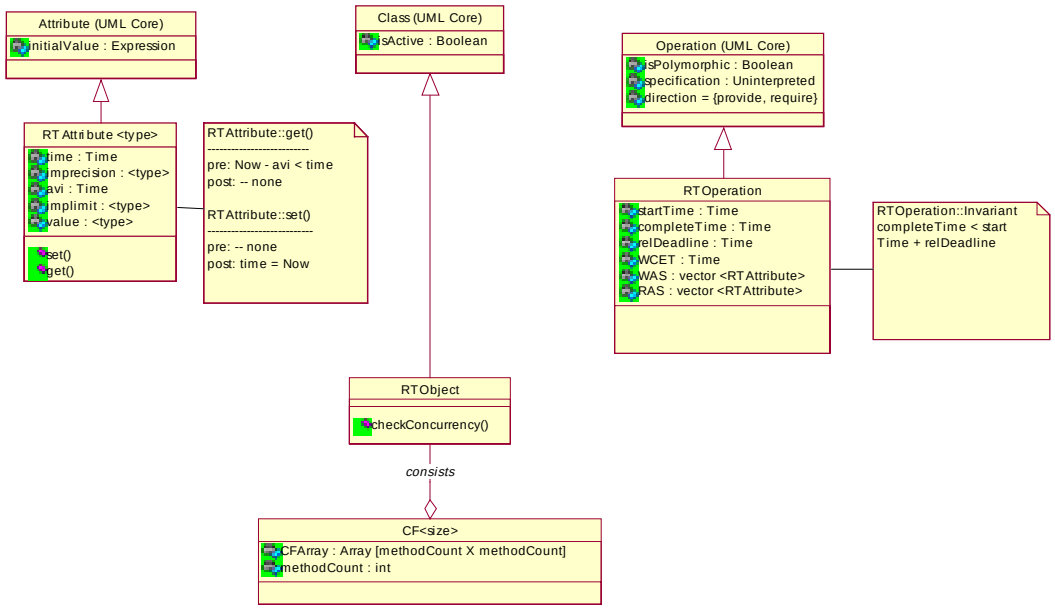


Figure 5 - RT-Object Package Classes

3.2 Metamodel Constructs

Figure 5 displays the new constructs that we have defined to model real-time objects. It also shows the existing constructs from the UML metamodel Core Package that we have extended. Recall Figure 3 from Section 2 that displays the Core package class structure. The following are brief descriptions of the classes of the Core package that we have extended.

3.2.1 Class.

A *Class* is a description of a set of objects that share the same attributes, operations, methods, relationships and semantics. Each object that instantiates a class contains its own set of values corresponding to the *StructuralFeatures* (attributes, methods, operations) in the *Class*. In the metamodel, *Class* is a child of *Classifier*. A *Classifier* is a model element that describes behavioral and structural features. It declares a collection of features such as *Attributes*, *Methods* and *Operations*. Thus, every class also has these features associated with it.

3.2.2 Attribute.

An *Attribute* is a named piece of the declared state of a *Classifier*. It specifies the range of values that the classifier may hold. An *Attribute* can specify an initial value that it will hold upon initialization. Each class can be associated with several attributes, that together describe the state of the objects defined by the class.

3.2.3 Operation.

An *Operation* is a *BehavioralFeature* that can be applied to the instances of a *Classifier*. It represents a service provided to change the state of an instance of a *Classifier*. The *Operation* has a signature that specifies the format of the parameters along with possible return values. A *Method* is an element in the UML metamodel that is related to an *Operation*. While an *Operation* is a conceptual construct, a *Method* is the implementation of an operation. The *Method* specifies the algorithms and techniques required to implement the *Operation* specified in the signature.

In the next section we describe how we have extended the above constructs to represent the functionality of real-time objects.

3.3 Real-Time Object Constructs

The real-time object model that we specify here formally expresses the RTSORAC object model described in Section 2.3 using UML constructs. The main construct of our model is the *RTObject* specification. In this section we describe the features and functionality of the *RTObject* class and the other constructs that comprise the model.

3.3.1 RTObject.

The *RTObject* construct extends the UML *Class* construct, and thus has the ability to contain *StructuralFeatures* like *Attributes*, as well as *BehavioralFeatures* like *Operations* and *Methods*. In our model, a *RTObject* can additionally contain real-time attributes (*RTAttribute*), and real-time operations (*RTOperation*) because these constructs extend the *Attribute* and *Operation* constructs respectively. The *RTObject* contains a compatibility function (*CF*) class as well to specify allowable concurrency among methods. These constructs correspond to the features of an object in the RTSORAC model. Notice that there is no construct to represent the set of constraints on the object. In UML, constraints are expressed textually in OCL. We express some general real-time constraints on the constructs that we have defined. In particular real-time applications, the specific constraints on the objects involved can be expressed using OCL in the appropriate places.

The *checkConcurrency* method on the *RTObject* provides the ability to check the compatibility of an invoked method with currently active methods. This can be used by concurrency control techniques that allow fine-grained, method-level locking.

3.3.2 RTAttribute.

The *RTAttribute* class extends the UML metamodel *Attribute* class. It includes the three components of the RTSORAC attribute: value, time, and imprecision. The *RTAttribute* expresses a parameterized type that represents the type of the value of the attribute. The time element expresses the time at which the attribute

was last updated. Its type is the Time type expressed in the UML standard. The imprecision element represents the amount of imprecision that may have been introduced into the attribute due to imprecise computation or relaxed serializability [11]. The imprecision element has the same type as the value because it represents how far from the "real" value the *RTAttribute* could be.

The *avi* element of the *RTAttribute* is its absolute validity interval [17]. This represents the maximum age allowed for the value of the attribute. Similarly, the *implimit* element expresses the maximum amount of imprecision that may be introduced into the attribute's value. These elements provide a mechanism for expressing certain standard constraints on the attribute.

The *RTAttribute* has two standard methods to *set* the value of the attribute and to *get* the value of the attribute. Several constraints are expressed on these methods. The precondition on the *get()* method expresses that the time field should be greater than the difference between the current time and the absolute validity interval. In other words, the age of the attribute should not be greater than the expressed maximum age (*avi*). A similar precondition can be expressed on the imprecision limit element (*Imprecision* < *ImpLimit*) if the application requires that imprecision be bounded. The postcondition on the *set()* method states that after the method executes, the time field should be equal to the current time. This indicates that the time field should be updated with the current time whenever the attribute's value is updated. Notice that UML does not deal with the enforcement of these constraints. Enforcement is left to the implementation of the system.

3.3.3 RTOperation.

The *RTOperation* class of the RT-Object package extends the *Operation* class of the UML metamodel. This allows a *RTObject* to be associated with a set of *RTOperations*, through the *Classifier*'s association with *Operation*. Additionally, the *RTOperation* expresses several timing characteristics. The *RTOperation* can express a relative deadline (*relDeadline*) in case its semantics involve a deadline separate from the deadline of the calling object. The *WCET* represents the worst case execution time of the operation. This can be used in scheduling analysis for scheduling and concurrency control mechanisms. The *RTOperation* specifies a read affected set (RAS) and a write affected set (WAS). Recall from Section 2.3.2 that these sets represent the attributes that the *RTOperation* reads and writes respectively. They can be used to allow fine-grained locking in concurrency control.

The *startTime* and *completeTime* characteristics represent the start time of the *RTOperation* and the completion time respectively. A constraint associated with the *RTOperation* class specifies an invariant expressing: *completeTime* < *startTime* + *relDeadline*. In other words, the execution of the *RTOperation* should not take more than the time expressed by the relative deadline.

3.3.4 CF

The compatibility function class (*CF*) is a new class defined to be part of the *RTObject* class. It represents a construct for allowing method level locking. CF contains a *methodCount* attribute that stores the number of methods associated with the *RTObject*. The *CFArray* is a *methodCount X methodCount* array of Boolean expressions that specify whether or not the corresponding pair of methods may execute concurrently according to the adopted concurrency control algorithm. The affected sets of the *RTOperation* can be used to determine this compatibility.

4 An Example – Real-Time Multi-User Virtual Environment

To illustrate how the RT-Object package can be used in designing real-time systems, we present an example application in which real-time objects are represented. A Real-Time Multi-User Virtual Environment (RTMUVE) is a real-time system in which widely distributed users collaborate to meet common goals and negotiate to come to agreement on possibly conflicting goals, all under timing constraints. The shared environment provides mechanisms for users to communicate using text, voice, and applications such as whiteboards. The users also share data such as documents, as well as real-time information.

We have implemented a prototype RTMUVE in which real-time objects are designed using the RT-Object UML package. In this section we describe our RTMUVE prototype and show how real-time objects play an integral part in the application. We then present a specific example to illustrate how real-time objects can be designed based on our UML *RT-Object* package.

4.1 RTMUVE Architecture

Figure 6 illustrates the architecture of our RTMUVE prototype. Parts of the prototype have been fully developed, while others are preliminary. The underlying architecture is based on the C2MUVE system developed by SAIC and ONR [18]. The RTMUVE server provides shared access of real-time and non-real-time objects to collaborating users. Within the RTMUVE server, users can hold virtual meetings, share data, chat, send messages, and communication in other ways in order to make time-critical decisions. The MUVE factory is where objects are created from various sources and then placed into the RTMUVE server to be shared by users.

4.1.1 Real-Time CORBA.

The communication among clients and the RTMUVE server is based on a dynamic real-time CORBA architecture as required in the Realtime CORBA Dynamic Scheduling RFP [19]. This allows clients to express timing constraints on requests to the RTMUVE server. The real-time CORBA middleware enforces the expressed timing constraints and provides a level of guarantee on the timliness of the responses to the requests.

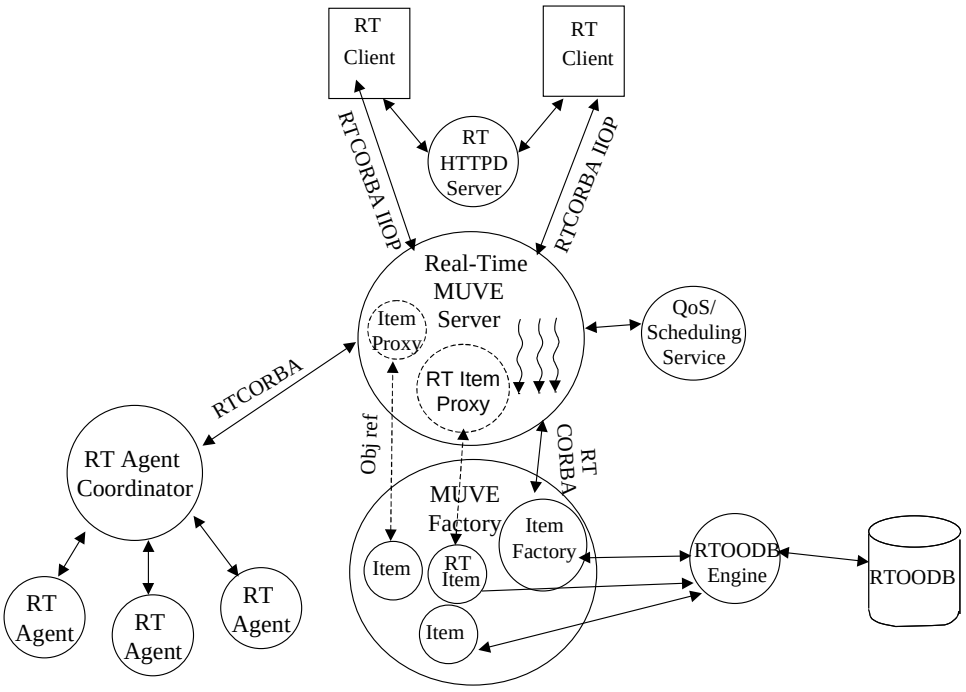


Figure 6 - RTMUVE Architecture

4.1.2 Real-Time Agents.

Also connected to the RTMUVE server through real-time CORBA is a system of coordinating real-time agents. These agents act autonomously on behalf of cooperating users, and other processes in order to aid the decision-making process under timing constraints. The real-time agents will often access the real-time data in the system in order to make decisions. For example, real-time agents can be used to filter data for the use of a particular user, within the expressed timing constraints.

4.1.3 Real-Time Objects.

The RTMUVE server contains all of the active objects in the multi-user environment. These objects include users, documents, applications, and real-time objects. The real-time objects have timing constraints that express how often they should be updated, and therefore require real-time scheduling within the RTMUVE server. The real-time objects in the RTMUVE architecture are designed based on the *RT-Object* UML package [20].

The real-time objects in the RTMUVE system use affected set semantics [11] for object concurrency control. That is, the compatibility function specifies that methods can execute concurrently if they do not both write the same attributes, and one does not read an attribute that the other one writes. When a method is called, a check of this compatibility is made before allowing the method to execute.

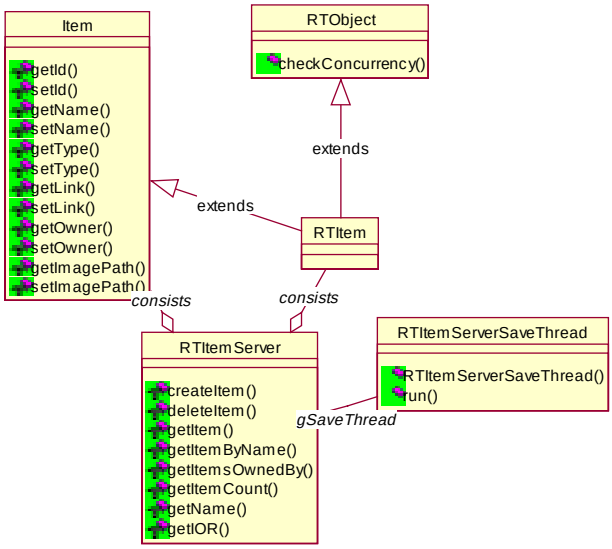


Figure 7 - RTItems in RTMUVE Design

4.1.4 Real-Time Object-Oriented Database.

The real-time objects in the RTMUVE system require time-constrained updates and some level of guarantee that they closely represent the state of the real-world. A real-time object-oriented database (RTOODB) is used in the RTMUVE architecture to provide the timely data to the system. The RTOODB is a main-memory database whose data is updated by sensors, and other environmental sources [21]. When users, agents, or other processes request real-time data in the RTMUVE, the server accesses the RTOODB and the real-time objects are updated from the RTOODB according to their respective timing constraints.

4.2 Real-Time Items

In the RTMUVE server, all objects are represented as *Items*. An *Item* in the RTMUVE contains not only all of the information associated with the particular object it represents, it also contains the relevant information for maintaining the *Item* in the RTMUVE server. This information includes the visual representation of the object (ImagePath) in the RTMUVE system, and the owner of the item. A real-time *Item* (*RTItem*) is an extension of both the *Item* class defined to support the features of items in the RTMUVE, and of the *RTObject* class defined to support the real-time features of the object.

The RTMUVE also has a class called *ItemServer*. The *ItemServer* is responsible for inserting and deleting specified items in the RTMUVE server. It also has a thread to periodically persist the items to a database. The non-real-time *Items* are persisted to a non-real-time database. In our prototype this database is MySQL [22]. The *RTItems* are persisted to the RTOODB based on their individual timing constraints. Figure 7 is a UML class diagram displaying the relationships among the *ItemServer*, the *Items*, the *RTItems*, and the *RTObjects.*

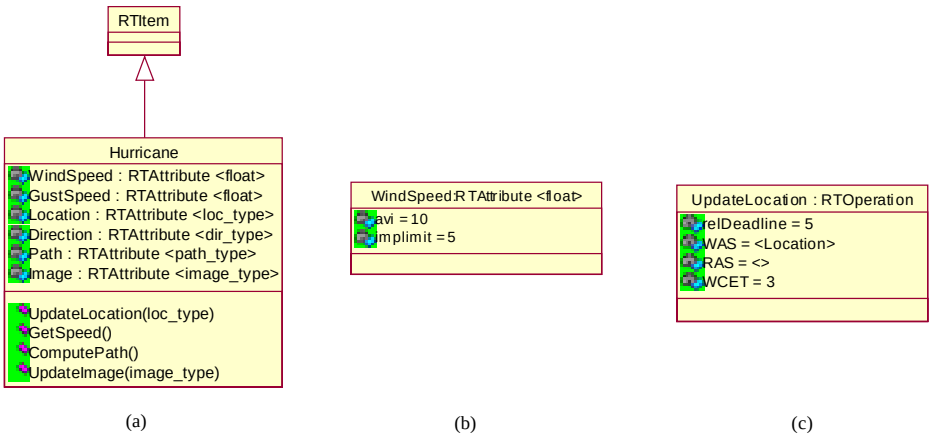


Figure 8 - UML Hurricane Object Design

4.3 Hurricane Example

The specific application of the RTMUVE system that we present here is a crisis action planning scenario in which a hurricane is heading towards the east coast of the United States. The planning involved in minimizing casualties and property damage in such a scenario requires collaboration among various groups including the National Weather Service, local police and fire departments, the National Guard, local television and radio stations, etc. In such a collaborative environment, the distributed users may share information such as planned evacuation routes, population reports and up-to-date satellite data. Some of this information, the satellite data for instance, should be updated periodically, and will therefore require real-time scheduling of its updates in order to provide the most accurate information to the users.

The kinds of decisions that have to be made in this application include whom to evacuate, which evacuation routes to follow, what flood prevention measures are required, and how and when to notify residents with information. All of these critical decisions require timely access to real-time data. For example, a hurricane can be represented with a satellite image combined with other data such as wind speed, gust speed, location and temperature. This kind of information has timing constraints on how often it should be updated. It also requires fine-grained concurrency control to allow multiple users to access it while still updating it with the latest data.

As an example real-time object in this application, consider a hurricane object, similar to the one described in Section 2.3. Figure 8a displays the UML class that represents the hurricane object. *Hurricane* is a subclass of *RTItem* so that it has features of both the *Item* class, and the *RTObject* class. Each of the attributes in the *Hurricane* class is specified as an instance of *RTAttribute* with a type specified because *RTAttribute* has a parameterized type for the value of the attribute. The operations of the *Hurricane* class are instances of the *RTOperation* class. This is not straightforward to express within the class diagram because the type of an operation represents the return value of the operation.

Figures 8b and 8c provide more information about the attributes and operations of the Hurricane class. These diagrams are object diagrams. Figure 8b represents the *WindSpeed* attribute of the *Hurricane* class – an instance of *RTAttribute*. It sets the value of the inherited attribute *avi* to 10. This specifies that the age of the *WindSpeed* attribute cannot be more than 10 seconds old. Similarly, Figure 8b expresses an object diagram that gives specific values to inherited attributes in the *UpdateLocation* instance of *RTOperation*. Each of the *RTAttributes* and *RTOperations* in the *Hurricane* class can be expressed in object diagrams in order to specify values for the attributes inherited from the RT-Object package classes.

In the RTMUVE system, an instance of the Hurricane class is persisted in the RTOODB and brought into the RTMUVE server by the *ItemServer* when a user, agent, or other process requests access to it. In the collaborative environment, users can view the satellite image stored as part of the object. They can also get information about the wind speed, gust speed, location, etc. that is available.

This example illustrates that when real-time system designers have to specify real-time objects they do not have to express the features of the object that are common to most real-time objects. These are included in the RT-Object package. The designers can concentrate on the specific features of the application.

5 Conclusion

This paper has presented a formal model of real-time objects through the definition of the UML package, RT-Object. The classes defined in the RT-Object package extend some of the most basic classes in the UML metamodel to provide a mechanism for expressing timing characteristics on data. These characteristics were first defined in the RTSORAC model and include representation of the age of an attribute, temporal consistency contraints expressed on attributes, imprecision in attributes, and imprecision constraints on attributes. The RT-Object class also provides a compatibility function and the expression of a real-time operation's read affected sets and write affected sets. These together provide a mechanism to support fine-grained, method-level concurrency control in real-time objects. Such fine-grained control can allow for a higher degree of concurrency and thus provide the potential for the system to meet more timing constraints [11].

We have illustrated the utility of the RT-Object package by describing its application in the RTMUVE design. Real-time objects play an essential role in the collaborative environment provided by the RTMUVE. Users of this real-time system expect that the data is current, and accurate. By importing the RT-Object package, the design of real-time objects in the RTMUVE application is simplified because the fundamental real-time features are already provided. The designers need only express the specific features of the application objects.

The OMG's RFP for a UML Profile for Scheduling, Performance, and Time [3] is a major step towards providing standardized mechanisms for specifying real-time systems. Our design and model of real-time objects will enhance the ability to express timing characteristics on the entire system.

The hurricane example that we have presented in this paper is a proof-of-concept example. In future work, we will apply the RTMUVE system to military collaborative applications in which surface ships and subsurface ships share data in battle management scenarios. This kind of application will provide further insight into the design features required by real-time objects.

References

[1] OMG. *Unified Modeling Language – V1.3 alpha R5.* OMG, Inc., March 1999.

[2] OMG. *UML Summary – V1.1*. OMG, Inc., Sept. 1997. OMG document ad/97-08-03.

[3] OMG. *UML Profile for Scheduling, Performance, and Time – Request for Proposal*. OMG, Inc., April 1999. OMG document ad/99-03-13.

[4] Mark Richters, Martin Gogolla. On Formalizing the UML Object Constraint Language OCL. *ER '98, 17th International Conference on Conceptual Modeling, Singapore*, November 16-19, 1998, 449-464. *Lecture Notes in Computer Science*, Vol. 1507, Springer.

[5] OMG. *Object Constraint Language Specification*. OMG, Inc., Sept. 1997. OMG document ad/97-08-08.

[6] G.Raghavan and M.M.Larrondo-Petrie. *A Formal UML Package for Specifying Real-Time System Constraints*. Florida Atlantic University Technical Report TR-CSE-99-22, April 1999.

[7] Bran Selic. Animated Structures: Real-Time, Objects, and the UML – Keynote Address. *19th IEEE Real-Time Systems Symposium* (RTSS98), Madrid, Spain, December 2-4, 1998.

[8] Bruce P. Douglass. *Real-Time UML*. Addison-Wesley, Reading, MA, 1998.

[9] J. Prichard, L. C. DiPippo, J. Peckham, and V. F. Wolfe. RTSORAC: A real-time object-oriented database model. *The 5th International Conference on Database and Expert Systems Applications*, Sept. 1994.

[10] J. W. S. Liu, K. J. Lin, W. K. Shih, A. C. Yu, J. Y. Chung and W. Zhao, Algorithms for Scheduling Imprecise Computations, *Foundations of Real-Time Computing: Scheduling and Resource Management*, A. Van Tilborg and G. M. Koob, ed., Chapter 8, pp. 203-249, Kluwer Academic Publishers, 1991.

[11] L. Cingiser DiPippo, V. Fay Wolfe Object-Based Semantic Real-Time Concurrency Control with Bounded Imprecision *IEEE Transactions on Knowledge and Data Engineering*, vol. 9 no. 1. Feb. 1997.

[12] K. Ramamritham and C. Pu, A formal characterization of epsilon serializability, *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, October 1995.

[13] T.-W. Kuo and A. K. Mok, : A semantics-based protocol for real-time data access, *Proceedings of IEEE Real-Time Systems Symposium*, December 1993.

[14] M. Squadrito, L. Cingiser DiPippo, and V. Fay Wolfe. The Affected Set Priority Ceiling Protocol For Real-time Object-Oriented Databases. *Proceedings of the First International Workshop on Real-Time Databases*, March 1996.

[15] M. Squadrito, L. Esibov, L. DiPippo, V. F. Wolfe, G. Cooper, B. Thuraisingham, P. Krupp, M. Milligan, R. Johnston, R. Bethmangalkar. The Affected Set Priority Ceiling Protocols for Real-Time Object-Oriented Systems, *International Journal of Computer Systems Science and Engineering - Special Issue on Object-Oriented Real-Time Distributed Systems*, v. 14. no.4.July. 1999

[16] OMG. *Enhanced Views of Time Request for Proposal*. OMG Inc., Oct. 1998. OMG document orbos/98-10-04.

[17] K. Ramamritham, Real-time databases, *International Journal of Distributed and Parallel Databases*, vol. 1, no. 2, 1993.

[18] SAIC. www.saic.com.

[19] OMG. *Realtime CORBA Dynamic Scheduling Request for Proposals.* OMG Inc., Jan. 1999. OMG document orbos/99-01-17.

[20] Ling Ma. *Integration of Real-Time Objects into a Distributed Multi-User Virtual Environment.* University of Rhode Island Technical Report TR99-268. May 1999.

[21] V. F. Wolfe, J. J. Prichard, L. DiPippo, J. Black. The RTSORAC Real-Time Object-Oriented Database Prototype. *Real-Time Database Systems: Issues and Applications*, Kluwer Academic Press, Dec. 1997.

[22] MySQL. www.mysql.com.