Constraint-Oriented Object Behavior Specification in Real-Time Control

D. Zmaranda

G. Gabor

Department of Computer Science, University of Oradea

5 Armatei Romane street., Oradea

ROMANIA

Abstract: Object oriented approach seems natural for real-time control systems that typically consist of sensors and actuators entities and control entities between these. Reusability and maintainability are, among others, positive attributes of object-oriented approach, because of their capability of abstraction, information hiding and inheritance. But, there are some drawbacks too: from specification level point of view, specifying the transformation of the system entities, with which timing constrains, should be associated lacks in means of expressing specific real-time aspects. This paper investigates which are the relevant aspects that should be considered when applying object-oriented issues to real-time systems development, emphasizing the improvements needed for specifying object behavior in a real-time control system. The identified concepts were illustrated using a simple and practical example.

Keywords: real-time control systems, object oriented decomposition methodologies, object states

Doina Zmaranda obtained her MSc in 1990 in Computer Science and Automation from Technical University of Timişoara, Romania, and PhD in Computer Science from Technical University of Timişoara, in the field of real-time computer control. Her current research interests include real-time systems and web programming techniques. She is currently employed at the University of Oradea, Department of Computer Science.

Gianina Gabor obtained her MSc in Computer Science and Automation in 1986 from Technical University of Timişoara and is expected to finalize her PhD thesis at the Technical University of Timişoara in Automatic Control field. Her current research interests include reliability of systems and modeling and simulation techniques. She is currently employed at the University of Oradea, Computer Science Department.

1. Introduction

Object-oriented programming offers many advantages for dealing with the complexity of today's real-time applications, including the ability to reuse software components. But object-oriented runtime executables have historically been large and slow to execute. This made object-oriented programming unattractive to real-time and embedded programmers. Now, new runtime packaging tools provide very compact and fast run-time executives, forcing real-time software engineers to abandon their "design-it-from-scratch" approach [2]. The new approach will be to build application software from existing components. The "virtual machines" existing now will be used to provide platform independence for object-oriented executables and to facilitate the re-use of existing software components.

Taking into accounts all the characteristics of the real-time systems like concurrency, timing constrains, safety, evolving nature, and so on, one can observe that the enterprise of developing a real-time application is a very difficult one. It seems to be clear that object-oriented techniques can make a significant contribution to the solution of difficult problems in this field. Among the reasons that support this argument one could mention [4]:

- object fit nicely with concurrency, since their logical autonomy makes them a natural unity for concurrent execution; this expands greatly our modeling power and enables the parallelism in the real world to be expressed in a more natural and easily understandable way
- the inheritance mechanisms which are provided with most of object-oriented methods support and encourage the reuse of already existing and tested software components, thus increasing the software quality

On the other side, the existing object-oriented approaches are still not the solution for the development of real-time applications, mainly because a mechanism to deal with timing semantics is still lacking [9]. Moreover, although many modeling CASE tools are available, tools for complex distributed real-time system design and implementation are scarce, and their timing semantics are still very poor [11]. In order to cope with the complexity and requirements of the real-time systems in the proper way, different object specification methodologies should be defined.

2. Specifying System's Object Structure Using Oriented Decomposition

Object orientation requires the decomposition be built around the data and the actions made on the data are encapsulated as its internal representation. In a real-time control system, these objects have attributes that describe the configuration and state of the objects: those that we know through instrumentation and those we know from some a-priori knowledge assumptions.

Generally, object oriented hierarchical decomposition supports [5]:

- abstraction, information hiding and encapsulation: an object is defined by the service it provides. The internal details are hidden and described in operation control procedures. The provided service is specified in the object's interface. The object's behavior is described in the object control structure
- hierarchical decomposition: parent objects may be decomposed into child objects
- control structuring: operations in objects are activated by control flows (threads): several threads may operate simultaneously in an object

From the structure point of view, we may look to a real-time system as being made up of the following separate (parallel) class hierarchies (Figure 1):

- the real-world environment we want to control - these classes captures all our a-priori knowledge and assumptions about the objects and processes we a trying to monitor and control
- the device used to acquire data about the controlled system - these could be an aggregation of measurement devices and control devices
- the processor environment which consists of both the knowledge about objects and processes derived from the measurement devices and from which we expect to derive control actions and the meta-class hierarchy of objects which describes the processor(s) in the problem domain and which models the execution behavior of these components

Each level of hierarchy forms associations with corresponding levels in other hierarchies [12]. Both the device space and the control processor space have abstractions corresponding to objects in the real-world hierarchy. Using the object-oriented approach, there is a close similarity among the object diagram and the physical structure of the real-time system, which results in many advantages, such as [10]:

- modularity: unrelated objects can be completely ignorant of each other, each object represent a self-contained piece of the problem which can be examined within the context of the system
- easy to understand: the abstract software objects represent entities in the physical world, reflecting the same semantics that are found in the problem domain
- easy to extend: every change in the real world physical system is easily identifiable, because every physical component that is important in the problem domain has a counterpart of the software side. Of course, sometimes more abstract objects that do not have a physical nature are needed (usually control objects). In spite of that, the physical structure of the real-time system is an excellent start point to build an object (class) diagram.

3. Specifying the Behavior of an Object

3.1. Defining the State of an Object

From the object-oriented point of view, the state of an object has different meanings: at the implementation level, the state refers to the values of the data fields of the object. During analysis and design, this term is used to indicate Boolean states in a state transition diagram. Generally, in object-oriented applications that don't have real-time requirements, proving the correctness is based on notion of states and the interpretation of programs as state transformers. From a mathematical sense, a state is simply a function that records a value for each object data at a given time.

States may be modified by actions that result during methods execution: each action corresponds to a transition from a state to another:

$$S_0 \rightarrow S_1 \qquad (1)$$

Whenever we have a sequence of actions a1…an, then starting from an initial state S_0, an object may have corresponding state transformation resulting in states $S_1 \ldots S_{n-1}$, as intermediary states and S_n as the final

state. Often the states S_0 and S_n are referred respectively as input and output states, and the program that results in the actions $a_1, \ldots a_n$ as a state transformer modifying the state S_0 into S_n.

In conventional object model, the state of the object is not taken into consideration when specifying the interface of an object. Using this approach, the main concern is to define the functionality of an object, by means of behavioral types. Behavioral type characterizes the behavioral properties of objects in terms of possible modification of its state [3]. But, in a real-time control system, an object might respond different to request of clients, depending on its state; thus, additional constrains should be specified for this category of systems.

Also, conventional modeling methods for real-time control systems impose the designer to specify deadlines for the system. Object oriented methods specify deadlines on the period between message reception and return from method execution. Other type of constrains imply a precedence order between the execution of methods, for example method B cannot start until method A is finished.

However, in object-oriented methodologies development there is no support for specifying real-time constrains on object states [4]. For example, the following requirement for a pump is difficult to be specified in existing methodologies: "the pump should not run more than 2 minutes when vibration condition appears". The lack to support for this type of real-time constrains is a problem of conventional approaches.

In conventional approach, finite state machines are used to describe the behavior of an object, by listing all possible states, the inputs (control or data) that move object from a state to another and the output resulting for each state. Finite state machines are usually represented graphically as state transition diagrams. When using finite state machines, the transformations performed are abstracted into states, implying that object perform a transformation when residing in a state. On the other hand, finite state machines provide some support for associating real-time constrains with states: transitions could be modeled with deadlines as a condition.

When a deadline is reached, the transition fires automatically and the machine goes to the next state. Unfortunately, none of these approaches provides a clear transition into an object-oriented language model [6]. Another practical problem related to finite state machines is that transitions are generally presented as atomic actions, within the state machine. This becomes a burden especially for real-time control systems, where some transitions could take significant period of time to be performed and, moreover, during this time other events might occur.

3.2. Identifying Relevant Aspects of the Behavior of an Object

As a consequence of the above problems, we might conclude that, for real time control systems objects, several aspects should be added when specifying object's behavior [1]. Thus, it is important to define the behavior of an object in response to the dynamics of its environment and its internal structure. The environment of an object consists of other objects that can interact with the object by sending messages. The internal structure of an object consists of nested objects that are possibly active and can send and receive messages. The encapsulating object contains, next to the nested objects, methods that are executed in response to received messages. This dynamics need to be constrained, if the object is to remain in a consistent state.

In conventional approaches, the behavior of an object is defined as the reaction of an object to the receipt of a message [5]. Object must search the method with the name that was used as a selector in the message. This behavior is considered unsatisfactory for real-time control systems where concurrency, real-time constrains and several other matters play an important role.

Several important aspects influence an object in a real-time control system. Each aspect describes the behavior of an object from a different point of view. These aspects are mainly de following [2]:

- **definition of method behavior** - this behavior includes the following important items:
  - definition of the region of object states that needs to be valid in order to execute the method
  - client object that are allowed to call that method
  - time and concurrency constrains for the method
- **definition of state behavior** – for each relevant state the following items must be specified:
  - the methods that can be executed
  - the clients that are allowed to the object in the current state
  - the time and concurrency constrains that are relevant for the state
- **definition of client behavior** – for each possible client, the following items must be defined:
  - object state region where the client can access the object
  - methods that are accessible
  - time and concurrency constrains for requests from the client

- **defining intra-object concurrency and synchronization constrains** – usually this is done using exclusion sets that define the methods that can only be executed mutually exclusive.
- **defining timing behavior** - with respect to the various deadlines in the system and the type of real-time constrains (hard and soft). For each deadline, the methods and object state regions for which the deadline is relevant, as well as the concurrency constrains must be specified.

This aspects can be used as a primarily decomposition in specifying object behavior for real-time control systems. Using this, the definition of object behavior is done not only in the one-dimensional state space, like in conventional approaches, but in the space formed by the five above dimensions:

Methods x States x Clients x Concurrency x Timing (2)

Using this approach, the basic object model could be expressed much formally starting from the idea that generally, an object contains nested object and methods. However, all user-defined objects, either directly or through nested objects, make finally use of primitive classes defined by the language (integer, boolean, char). Regarding the five dimensions described above, the state space of an object that contains n nested objects O_1, $O_2 \ldots O_n$, can be expressed as the cartesian product of the individual state space of each included object:

$$S_o = S_{o1} \times S_{o2} \times \ldots \times S_{on} \quad (3)$$

The domain of each state is not limited to boolean, so that a single state dimension can be then expressed as a much larger collection of states; for example, {1..20}

At any point during its existence, the object O is exactly at one point on its above-define state space. Moreover, let assume that object O has a method set:

$$M_o = \{M_{o,1}, M_{o,2}, \ldots M_{o,p}\}, \quad (4)$$

and, each method has a possibly empty set of arguments:

$$A_{Mo,i} = \{a_{Mo,i,1}, a_{Mo,i,2}, \ldots a_{Mo,i,q}\} \quad (5)$$

The behavior of a method $M_{o,i}$ can be defined as a function on the whole object space:

$$B_{Mo,i} : S_o \rightarrow S_o \quad (6)$$

By this, each method could be viewed as a vector in the state space of the object, because it takes the object from a particular state from S_o to another particular state from S_o.

A logical consequence of the several constrains defined above is that the object has a dynamic interface: clients of the object require information about the state of the server object in order to determine when the required interface element can be accessed. But, providing the clients with direct access to the state of the server would break encapsulation; thus, means for correct determining the state of an object by a client should be specified.

The main problem that arises regarding this matter is that, as it was shown above, the object state space is very large, so we will return to the problem of the explosion of states as for finite state machines. The idea for solving this problem is based on restricting the state space to only the significant ones, or so called relevant states. Of course, the designer should define the sub-domains from the state domain space for each relevant state and specific methods that return current relevant state should be provided for the clients.

Another problem, linked with the one above, related to the type and number of clients of the server object; thus, client categories have to be defined, and, for each message must be determined whether the sender of the message is a member of a specific client category. The message is then a subject to the behavioral relation(s) defined from that client category. Consequently, a relation between the object and a client category implies some behavioral constrains, such as:

- **client-based access** – restriction of access to interface elements based on client category: some clients are allowed to access certain interface elements, some are not
- **state-based access** – restrict the access of certain client categories when object is in a certain state
- **concurrency** - controls concurrent access of object by several clients: to maintain internally consistent state of the object, some client must access the server object mutually exclusive

- **real-time** – the interval within which a message returns after being received by the object is generally restricted by some upper limit (hard or soft deadlines). These constrains control real-time behavioral of the client-server relation

4. A practical Example: the CO_2 Tank Object

The geothermal power plant is a component of the cascaded geothermal energy utilization system, and is used to convert the energy of the geothermal water into electrical energy using CO_2 as working fluid. The elements of the power plant are the following: vaporizers (heat exchangers used to vaporize the CO_2), a reciprocating engine connected with the electric generator, a make-up and expansion CO_2 tank, condensers (heat exchangers used to condense the CO_2) and a CO_2 pump. Figure 2 shows the block scheme for the geothermal power plant [7].

The CO_2 tank can be filled and emptied; from the structural point of view, object decomposition of this component of the geothermal power plant system consists of the following items:

- the CO_2 tank, which includes of the following objects
- a level sensor – object O_{level} with specific state space S_{o_level}, defined in the range of possible levels, ex: {0.1 ... 1 m}
- a temperature sensor - object $O_{temperature}$ with specific state space $S_{o_temperature}$, defined in the range of possible temperatures, ex: {20…80° C}
- two valves, one used for filling and one for emptying the CO_2 tank respectively - object O_{fill_valve} and O_{empty_valve} with specific state space $S_{o_fill_valve}$, respectively $S_{o_empty_valve}$, defined in the range of {0..100 %}

When modeling this application using an object-oriented method, the system should be decomposed into several objects, both for physical and conceptual elements of the system. To illustrate the theoretical issues described above, an example using the CO_2 tank from the above scheme is given in Figure 3.

Based on the object classification presented in 2, we can identify the following categories of objects:

- one environment object: the CO_2 tank object
- four device objects: a temperature sensor and a level sensor (two measurement devices), an input and an output valve (two control devices). The state of CO_2 temperature sensor is represented using voltage levels. The level sensor contains, as a state, the liquid CO_2 height in some measurement unit (ex. m). Similarly, the state of the input and output valve is represented as an open percentage
- processor objects: there are no processor objects represented here; but they consists of the objects which process information provided by the measurement devices and initiate control actions to the control

From the behavior point of view, the state space of the CO_2 tank object is formed by composing the following component object states:

$$So_{}_{CO2_tank} = S_{o_level} \text{ x } S_{o_temperature} \text{ x } S_{o_fill_valve} \text{ x } S_{o_empty_valve} \quad (7)$$

But, if we examine the CO_2 tank object more closely, we can identify four relevant states: full, filling, emptying, minimum_level. For example, on can define that, when the level is in the range of {0.1…0.15 m} the CO_2 tank is in the "minimum_level" state; when level is in the range of {0.95 ... 1 m} it is in the "full" state; otherwise, the CO_2 tank is whether filling or emptying [8]. Figure 4 illustrates these relevant four states from the CO_2 tank object.

The CO_2 tank object must have the following methods: start_fill, start_emptying, stop_fill, stop_emptying, stop_all (in case of emergency). When trying to define the dynamic model for the CO_2 tank object, all types of constrains on tank behavior should be considered. These constrains are known from system specification, and could be divided into several groups:

- **state based constrains** – restrict the behavior if the object with respect to its state. For the CO_2 level object, these are the following:
  - the CO_2 tank should not be at minimum level for longer then 30 seconds
  - the CO_2 temperature in the tank should be between (26.4 … 26.6 °C)
  - the temperature in the tank should not be greater than 27 °C more than 10 seconds

- **client-based constrains** – restricts the behavior of the object with respect to the client object that request services. Basically, there are tree categories of clients for level object:
  - controller – is allowed to access all the methods and states of the CO_2 tank object, except the emergency stop_all method
  - observer – these objects are only allowed to read the state of the CO_2 tank: temperature, level, etc.
  - emergency – emergency objects are only allowed to execute stop_all method
- **concurrency-based constrains** – multiple client objects may access CO_2 tank object simultaneously. These accesses need to be synchronized for correct operation, otherwise, inconsistent object states may occur. Given the above categories of clients, concurrency can be specified by client category:
  - controller – requires mutual exclusion of controller threads
  - observer – can executes fully concurrent (only reading)
  - emergency – requires mutually exclusive access
- **real-time constrains** – restrict the time frame in which the object can respond to a message. Although, real-time constrains are primarily concerned with deadlines, but also the start time of a method execution can be specified. For the CO_2 tank example, the following real-time constrains could be specified:
  - controller – all methods have 1 second hard deadline
  - observer – for all states, 10 seconds soft deadline
  - emergency – stop_all: 0.1 seconds hard deadline

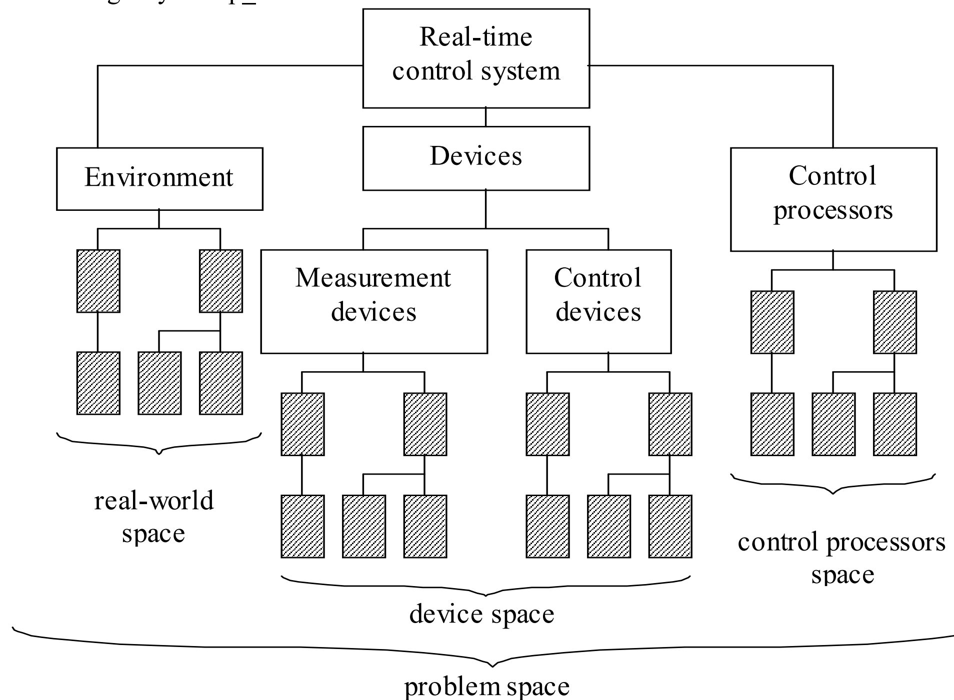


Figure 1: Real-time Object Hierarchies

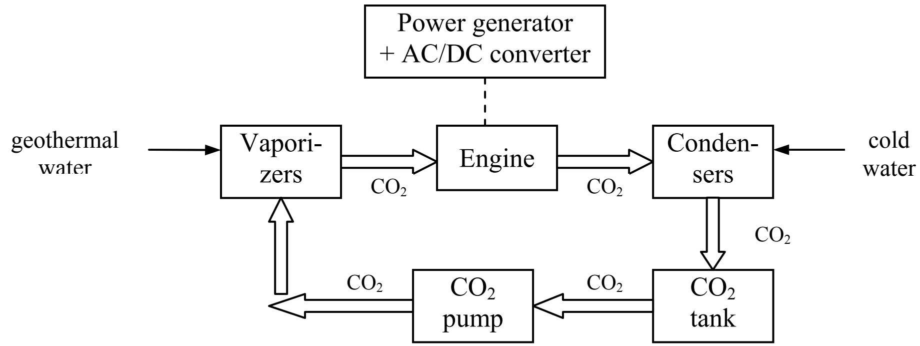
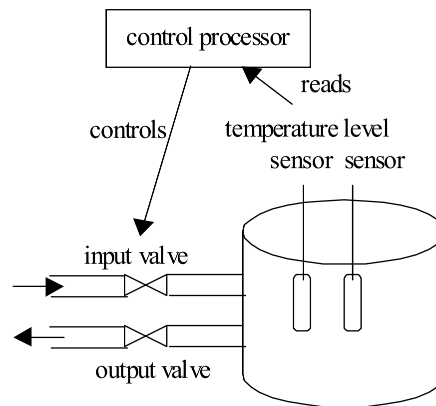
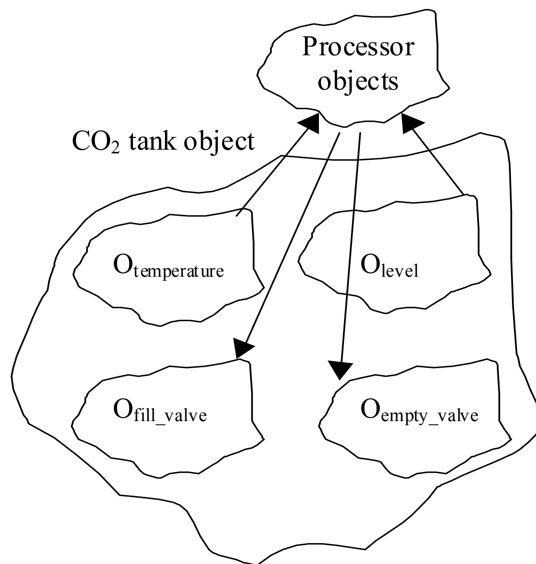


Figure 2: Geothermal Power Plant – Block Scheme



a) physical structure



b) conceptual structure

Figure 3: The CO_2 Tank Structure

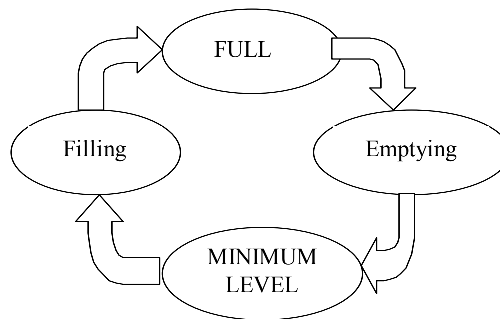


Figure 4: Four Relevant States for CO_2 Tank Object

5. Conclusions

In this paper problems related to the object states and finite state machine were discussed. A different approach for dealing with object states is presented. This approach tries to overcome some anomalies of the conventional object model regarding objects.

It was outlined that, from the real-time point of view, several aspects of objects must be considered: the most important is the one that in a real-time control system, objects must exhibit different interfaces to different clients and thus depending on object state.

REFERENCES

1. SVRCEK W., D. MAHONEY, B. YOUNG, **A Real-Time Approach to Process Control**, John Wiley & Sons, 2000
2. DOUGLAS B. P., **Doing Hard Time: Objects and Patterns in Real-Time Software Development**, Reading MA: Addison Wesley Longman, 1998
3. BOOCH G., **Coming of Age in an Object Oriented World**, IEEE Software, November 1998
4. COOLING J. E., **Real-Time Software Systems: An Introduction to Structured and Object Oriented Design**, PWS Publishing Company, 1997
5. ELIENS A., **Principles of Object-Oriented Software Development,** Addison-Wesley Inc., 1995
6. BENNETT S., **Real Time Computer Control**, Prentice Hall, 1994
7. GABOR G., **Reliability considerations on the control system of a geothermal power plant**, Proceedings of ECI'02, Kosice-Herlany, Slovakia, 2002, pp. 288
8. GABOR G, GAVRILESCU O., **Fault diagnosis in the control system of a geothermal power plant**, Proceedings of CONTI '02, Timişoara, Romania, 2002, pp.111
9. IRWIN G.W., **The Engineering of Complex Real Time Computer Control**, Kluwer Academic Publisher, 1996
10. KOPETZ, H., **Real-Time Systems - Design Principles for Distributed and Embedded Applications**, Kluwer Academic Publishers, 1997
11. VERILOG INC.,**Object GEODE - Object Oriented Real-Time Techniques. Method Guidelines**, 1996
12. ZMARANDA D., **Using Object-Oriented Paradigms During the Development of Real-Time Control Applications**, Proceedings of the EMES'03 International Conference, Baile-Felix, 2003, pp. 234