OBJECT-EVENT APPROACH TO MODELING TASKS

Alexander Shvets
shvets@cs.dgtu.donetsk.ua

Abstract

Shvets A.G. Object-event approach to modeling tasks. The existing approaches for building programming models are analyzed. The main components of object-event model are discussed. Main concepts for way how to realize object-event gear are described. This article outlines basic features of object in the object-event environment. Main components of modeling environment are explained. In the conclusion the main requirements to the object in the object-event model are formulated.

The analysis of the existing approaches

For modeling of real world processes researches used the power of the programming languages often. Various approaches for building programming model nowadays are known. The *object-oriented approach* is the last achievement of the modern programming. It has advantages in comparison with other approaches, such as the structural approach [1, 2] or approach on the basis of organization of the data streams [3]. The first approach doesn't support an ability of abstraction's extraction and data concealment, and second - is applicable only in case when the model contains direct links between input and output streams of modeled system [4].

However, introduction of object notion is only partial decision of the problem, because it does not express *interactions* between different objects, available in the model. Objects themselves aren't of special interest and the model's work is realized by specifying interaction between objects. Let's name the approach as *object-event* when it is taking into account not only insertion of objects into the model, but also and interaction between different objects. The purpose of this article is to present missing parts, addition of which to object-oriented approach will allow to have the completed system with the ability to create models on interaction-plus basis.

Such languages, as C++ and Object Pascal support the object-event approach only partially. In these languages the concept of object is realized, but concept of event and mechanism of interaction between objects is missed. Realizations of these languages for graphical environments, such as Windows, Macintosh or X-Window support the concept of event, however this support is tightly connected to a platform and is not the property of the language. It enables to assert that in these languages there is no complete support of object-event paradigm.

Object-event model

The following concepts are the basis of the object-event approach:

Object is some essence of the real or virtual world considered as a whole. The object can be considered as some aspect of a modeling reality [5].

Event is an object of a special kind, which transports an information from one object to other. The second name of event is *message*.

In general case objects can transfer to each other information of any kind. Using objects and events as base building components, it is possible to reduce any task of programming to an *object-event model*. Figure 1 contains the example of such model.

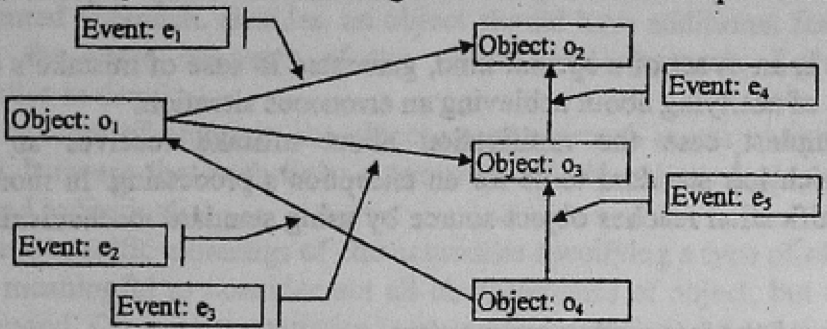

Figure 1. An example of simple object-event model

This model works in the following way. In the beginning of modeling user selects the object-initiator, which starts the process of modeling. For i-th interval of time the object-source sends some event to object-receiver. After getting of event the receiver should definitely react to it by executing some work associated with this event. The modeling stops after achievement by model some final state, in which further generation of events is not meaningful or is impossible.

Basically, any object can act as an initiator. However, model usually contains an additional component - the step generator. It creates events by certain rules, which are spreaded inside the model, realizing the life of model.

Components of the interaction mechanism

The event transmission from one object to another means, that between them there is an *interaction*. The object-source of event acts as a leading part – *transmitter*, and object-receiver - as passive part - *receiver*.

The interaction between the transmitter and the receiver is possible only after achieving of some condition. It means the following.

a) The transmitter should have a special body, capable to produce messages, necessary for the transmitter (*generator*). The generator has the ability to determine the moment when the generation of event is required.

b) The receiver should have a special filtering body, capable to perceive result of interaction (*sensor*). The sensor passes to the receiver only those events, on which it is configured.

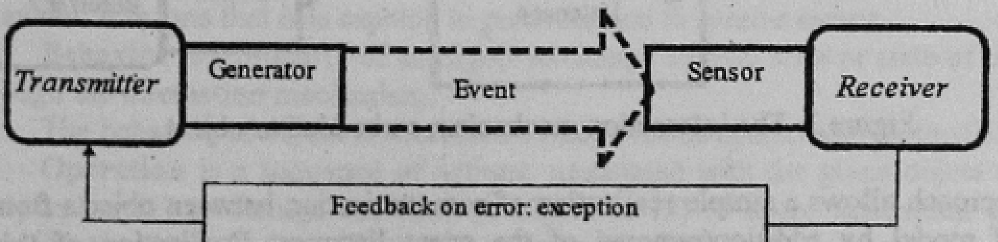

Figure 2. The basic components of the interaction mechanism

The object can act as a *transceiver*. In this case its structure contains both sensor and generator. If an erroneous situation arises during the processing of event in the object-receiver, this object should process a mistake or notify object-source about mistake.

Exception is an event of a special kind, generated in case of mistake's occurrence with the purpose of notifying about achieving an erroneous situation.

In the simplest case the notification about mistake receives an execution environment, which has standard tools for an exception's processing. In more complex situations the notification reaches object-source by using standard mechanisms of events transmission.

Realization of the interaction mechanism

The interaction mechanism can be realized at a level of execution environment or in the structure of object. Each of the offered approaches has advantages and lacks. In the first case the event should keep an additional information: the receiver's address. The transmitter generates event and sends it to the *system queue*. The handling program for system queue recognizes the recipient of this event and sends it to *recipient's queue*. The receiver scans its own event queue and if it isn't empty, perform its processing.

Such approach is simple, but it requires simultaneous start at least of three processes: for the generator, receiver and system queue. Besides, here is the non-trivial interpretation of event - placement of event in special "envelope" with the addressee.

In the second case the event doesn't contain any additional information and the interaction mechanism is embedded in object [6]. Thus the object-source supports the list of the registered objects wishing to receive the given event - *the listeners list.* During generation of the next event all listeners from the list are notified by an object-source (figure 3). *The generator* is realized as a cycle performing passage over listeners' list and activated *the sensor* of each receiver. The sensor is realized as some operation, containing a code-reaction on the input action. With such approach the knowledge of a method-reaction of object-receiver is required for object-source. For that it is possible to take advantage of a well-known design pattern Observer [6].

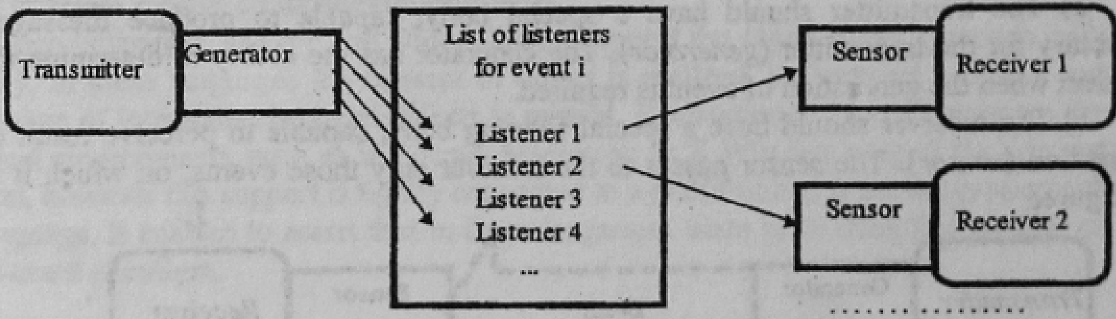

Figure 3. The interaction mechanism embedded to object

This approach allows a simple realization of communication between objects from the editor of model by addition/removal of the event listeners. Realization of this approach needs only one process - for object-source.

The main features of object in the object-event model

The object, which exists in object-event model, should have the certain set of characteristics. The part from these characteristics is not new and has its source in the object-oriented approach. Besides, an object should have additional features, allowing it to realize the basic concepts of the object-event approach. Let's consider these characteristics in detail.

The concept of object in the object-event approach meets to concept of the *finite automaton*. It means that both finite automaton and object have such properties, as state, behavior and individuality.

State is specific meanings of characteristics specifying a type of object.

It is meaningful to consider not all characteristics of object, but only those which can be changed. Other characteristics (constants) are excluded from the description of a state, because they can't change a state of object.

The object has the ability to change the state and this change is performed *directly*, not on the basis of the event dispatching mechanism. When the object contains other objects as its own components, its state is defined as the sum of the states of these components. The change of object's state will correspond to changes of states of its components.

Individuality is a property of object to differ from other objects of this type or any other type.

The individuality means, that we distinguish processes of creation of various objects (we perceive objects as individuals) and implicitly we assign them some unique descriptors in the creation period. The name of object or its allocation address in memory can act as a descriptor. In the first case there can be some problems, because the programming languages allow to have some names for one object (aliases' creation mechanism) or one name for group of objects (polymorphism). At the same time address of memory (or at least index descriptor in address table) allows to identify object in unique way.

If objects of one type have identical properties, such as a state or individuality, there can be the following relations. **Equality** is a relation between objects, showing that the considered objects have identical state. **Identity** is a relation between objects, showing that the considered objects represent the same individual.

Sometimes there is a requirement for creation of an exact copy of object, such that a state of the original and the copy are coincide. **Cloning** is an ability of object to create its copy with the same state. Source object and its clone are *equal* but are *not identical*.

The object also should have an ability to influence to other objects of model with the purpose to change their states or to change own state under influence of other objects of model. It means that it is capable to *generate* and to *receive* events.

Behavior is a property of an object to change its own state or state of other objects through the interaction mechanism.

The behavior of object is described as some list of operations (protocol).

Operation is a sequence of actions, associated with the given object and serving the purpose (1) to change the state of object or (2) to cause generation of event for changing the state of other objects. Other name for operation is *method*.

All operations can be divided on three groups:
- creation and destruction of object - *constructor* and *destructor*;
- reading and changing object's state - *selector* and *modifier (mutator)*;
- sending and receiving events - *generator* and *sensor*.

The object during the life passes through phases of creation, existence and destruction. These phases has proceed not isolated but in a context of the program execution. The program as well as object has the beginning and the end. Existence of object and process of the program execution in general case are the independent phenomena and shouldn't affect each other by no means. However in the traditional programming languages it is not true and area of object existence is limited between the beginning and the end of the program. Here we have the collision.

Stability is a property of object to exist in time irrespective of the process that has created this object.

For stability object should be able to save and to restore its own state.

Serialization is a property of object to save the state as a sequence of the primitive data.

Deserialization is a property of object to restore the state from a sequence of previously saved primitive data.

Objects also should have an ability to exist and to demonstrate the behavior simultaneously and independently from each other. It requires from execution environment supporting of parallel execution of operations. Parallelism in a case of object-event model means that the object is considered as a *process*.

Process is an object that allows for some own operation to be executed in parallel with operations of other objects-processes.

The object that realizes the concept of process refers to as *active*, otherwise *passive*.

In single-threaded systems there can be simultaneously only one active object, while in parallel systems there are a set of such objects.

Not all components of model should be active. For example, events and exceptions represent only containers for certain information. After transfer of event this information can be saved in the receiver or transferred further to other objects and event can be destroyed.

We can present the basic components of the interaction mechanism (fig. 2) as processes. The separate processes will correspond to the transmitter and to the receiver. The generator and sensor will be realized as operations of processes that are performed in parallel.

Because the active objects can perform their operations simultaneously with operations of other objects, there can be cases of asynchronous access to a state of the object. If one active object changes a state of a shared resource and in this time other one reads this state, there can be races. It means that the object reading a state of a shared resource will receive incorrect value. Therefore it is necessary to have a means forbidding simultaneous access to a shared resource.

Monitor is a property of object to give the state only to one object simultaneously.

It is said that the access to a resource enclosed in the monitor is *synchronized*. It can cause a large overhead. This overhead can be reduced if to work not with a resource, but with its clone. The creation of a clone is located in the monitor and work with a clone is excluded from the monitor.

Modeling environment

During its existence the model can be in a state of development or in an execution state. In the first case it should have mechanism of interaction with the developer of model and in the second - with the user (figure 4).

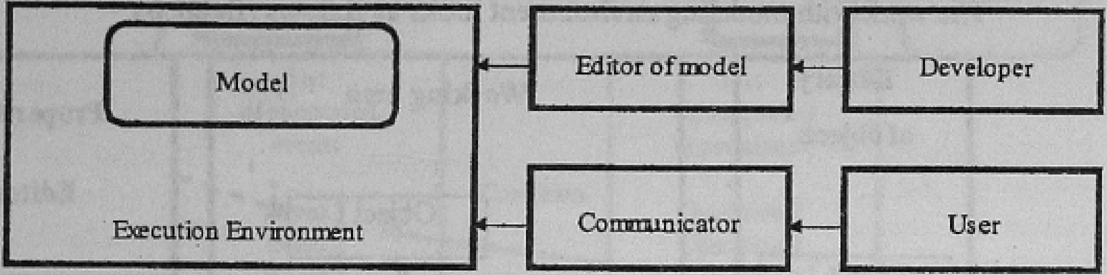

Figure 4. Modeling environment

The editor of model provides creation and updating of model. It can be missing in the system, in this case developer realizes model manually by writing of the appropriate program code. Execution environment with the editor of model refers to as *the development environment.*

The communicator gives tools for interaction of the user with execution environment with the purpose to control the model's work. Communication tools can be realized in the following form.

Interaction on the basis of files. The communicator is missing. Process of the user interaction with execution environment is considered as a process of data input to a input file and reading of modeling results from an output file. The model can be invisible or can contain primitive abilities for visualization of modeling process. The model configuration is possible only before the start of modeling process by using tools which can specify the appropriate parameters of model in the input file.

Interaction in a command line mode. Communicator is realized as the simple textual interpreter *embedded into model.* The process of interaction is represented as a process of sending commands from user to model. The model responds to them by changing the state or returning the requested information to the user. The model can be visible or invisible. The model configuration is possible at any moment of time with the help of the interpreter's commands. Example – shells for Unix kernel.

This mode is easy to understand because the *attention* of the user is attached to one point – point of input of the next command. The basic lack is that the user should hold in a head or in a hand all commands of the system, which describe interaction between the user and execution environment.

Interaction in the object-event form. The editor of model and communicator are standard components of execution environment – they are built-in into execution environment. Therefore model doesn't contain anything extra. The interaction of the user with the modeling environment is realized on the basis of *a visual mode,* which allows free manipulation of objects composing the model. Such visual environments as Delphi, Power Builder and Symantec Visual Cafe support this form of interaction.

Environment can be represented with the following structure:

- **Library of objects** serves for a storage of ready objects used as a "building material" during the model construction;
- **Working area** is an area for model's construction;
- **Editor of properties** allows to setup objects of model in the working area;

- **Editor of interactions** allows to specify how different objects cooperate with each other by using events dispatching mechanism.

The work with modeling environment looks as follows (figure 5).

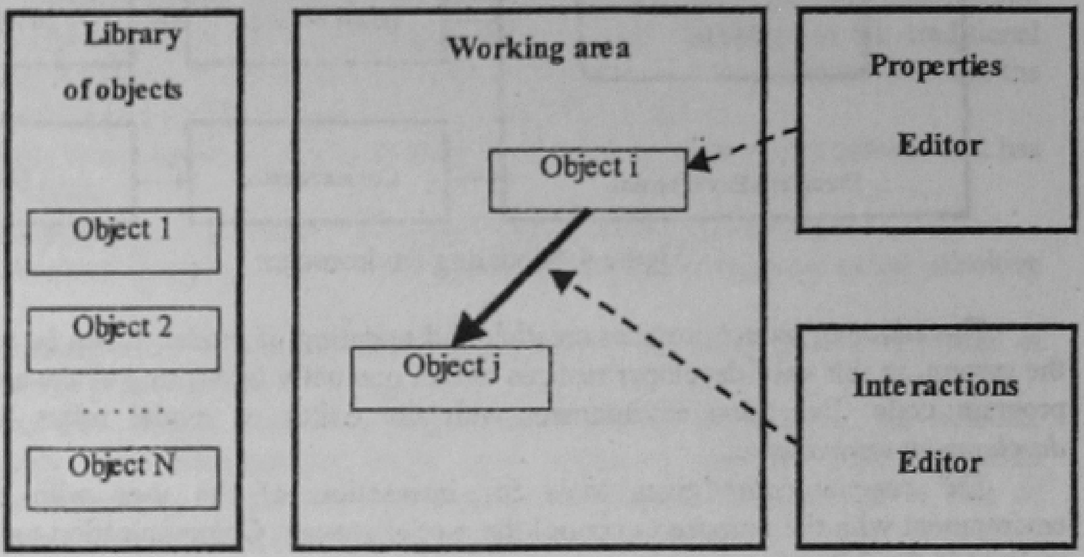

Figure 5. The process of work with modeling environment

- The developer chooses the next object from the *library of objects* and places it on the *working area* in the necessary place. This operation is performed so much times, how many objects are required for the realization of a task.

- In turn the objects on the working area are selected and their properties are customized in the *editor of properties*;

- For each pair of objects, between which there should be an interaction, the connection is established with the help of the *editor of interactions*.

- The object-generator of steps is added to model. It acts as an initiator of modeling process.

- The model is started on execution by transferring of control to the modeling initiator.

The interaction between objects is performed under the following scenario:

- The object-transmitter is selected. It acts as the *initiator* of communication and can generate some event.

- From the *list of supported events* for object-transmitter the desired event is selected.

- The condition of event generation is defined.

- The object-receiver is selected.

The operation-reaction to incoming event is selected from *list of supported operations* for object-receiver.

The list of events supported by object is formed on the basis of events, which can produce its generator and the list of operations, supported by object - on the basis of events, which can perceive its sensor. On figure 6 the process of communication's creation between transmitter and receiver is shown. The visual model created in the working area can be saved for its using in the future.

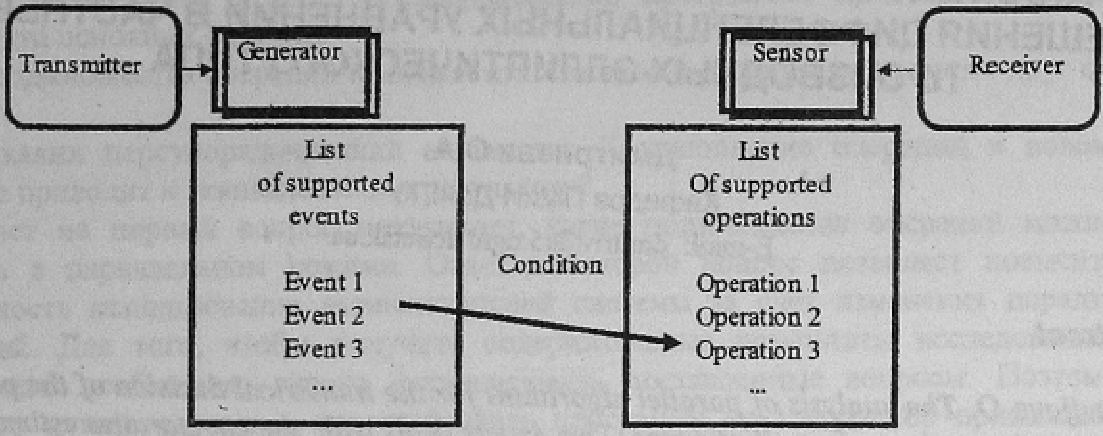

Figure 6. Process of interaction creation

The conclusion

After object-oriented era we see new horizons – object-event point of view. It expands the object-oriented approach, emphasizes the attention of developers to the mechanism of information interchange between different objects.

This approach is more effective, because it produces more readable sources – consequently – more robust code that is easier in support. Concept of event is well known for human, so programs with events are easy to understand.

It is possible to determine the basic properties of an object, existing inside the object-event execution environment:

- it has a *state;*
- it has certain *properties*;
- it exhibits *the individuality*;
- it displays the *behavior* by catching and generating events;
- it displays a *stability*, independence from the parent program;
- it is an *active* component, i.e. process;
- it can be *customized* by using development environment;
- it can *interact* with other objects by a sending of appropriate events.

The literature

1. Wirth, N. 1986. Algorithms and Data Structures. Engelwood Cliffs, NJ: Prentice-Hall.

2. Dahl, O., Dijkstra, E., and Hoare, C. A. R. 1972. Structured Programming. London. England: Acadfemic Press.

3. Jacson, M. 1975. Principles of Program Design. Orlando, FL: Academic Press.

4. Grady Booch. Object-Oriented Analysis and Design with Applications. Benjamin/Cummings, Redwood City, CA, 1994. Second Edition.

5. Wegner, P. June 1981. The Ada Programming Language and Environment. Unpublished draft.

6. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA, 1995.