ACTA TECHNICA CORVINIENSIS
BULLETIN OF ENGINEERING

[1.] ANCA IORDAN, [2.] MANUELA PĂNOIU

DESIGN OF SEQUENCE DIAGRAMS FOR IMPLEMENTATION OF A DYNAMICAL SOFTWARE FOR DOING GEOMETRICAL CONSTRUCTIONS

Abstract:
This paper presents a software package, which can be used as educational software. The informatics system, including modern methods and techniques, will lead the subject which is using it to gain experience in understanding and managing the knowledge from geometry field and will offer the comfortable and efficient access to the newest information and knowledge. The investigation can be oriented towards reaching of some precise purposes or can be an exploration.

Keywords:
UML, Sequence Diagram, Educational Software

INTRODUCTION

Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of software engineering [1]. UML includes a set of graphical notation techniques to create abstract models of specific systems.
The Unified Modeling Language (UML) is an open method used to specify, visualize, construct and document the artifacts of an object-oriented software-intensive system under development. UML offers a standard way to write a system's blueprints, including conceptual components such as: actors, business processes, system's components and activities, as well as concrete things such as: programming language statements, database schemas and reusable software components.
UML combines the best practice from data modeling concepts such as entity relationship diagrams, business modeling (work flow), object modeling and component modeling. It can be used with all processes, throughout the software development life cycle, and across different implementation technologies. UML has succeeded the concepts of the Booch method, the Object-modeling technique (OMT) and Object-oriented software engineering (OOSE) by fusing them into a single, common and widely usable modeling language. UML aims to be a standard modeling language which can model concurrent and distributed systems. UML is not an industry standard, but is taking shape under the auspices of the Object Management Group (OMG). OMG has initially called for information on object-oriented methodologies, that might create a rigorous software modeling language. Many industry leaders have responded in earnest to help create the standard.
UML models may be automatically transformed to other representations by means of QVT-like transformation languages, supported by the OMG. UML is extensible, offering the following

mechanisms for customization: profiles and stereotype.
UML is not a development method by itself, however, it was designed to be compatible with the leading object-oriented software development methods of its time. Since UML has evolved, some of these methods have been recast to take advantage of the new notations (for example OMT), and new methods have been created based on UML. The best known is IBM Rational Unified Process (RUP). There are many other UML-based methods like Abstraction Method, Dynamic Systems Development Method, and others, designed to provide more specific solutions, or achieve different objectives.
It is very important to distinguish between the UML model and the set of diagrams of a system. A diagram is a partial graphical representation of a system's model. The model also contains a "semantic backplane" — documentation such as written use cases that drive the model elements and diagrams.
UML diagrams represent two different views of a system model [2]:

- *Static view: Emphasizes the static structure of the system using objects, attributes, operations and relationships. The structural view includes class diagrams and composite structure diagrams.*
- *Dynamic view: Emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. This view includes sequence diagrams, activity diagrams and state machine diagrams.*

UML models can be exchanged among

SEQUENCE DIAGRAMS

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order [3]. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called Event-trace diagrams, event scenarios, and timing diagrams.
A sequence diagram shows, as parallel vertical lines, different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.
The UML 2.0 Sequence Diagram supports similar notation to the UML 1.x Sequence Diagram with added support for modeling variations to the standard flow of events.
If the lifeline is that of an object, it is underlined. Note that leaving the instance name blank can represent anonymous and unnamed instances.
In order to display interaction, messages are used. These are horizontal arrows with the message name written above them. Solid arrows with full heads are synchronous calls, solid arrows with stick heads are asynchronous calls and dashed arrows with stick heads are return messages. This definition is true as of UML 2, considerably different from UML 1.x.
Activation boxes, or method-call boxes, are opaque rectangles drawn on top of lifelines to represent that processes are being performed in response to the message (Execution Specifications in UML). Objects calling methods on themselves use messages and add new activation boxes on top of any others to indicate a further level of processing.
When an object is destroyed, an X is drawn on top of the lifeline, and the dashed line ceases to be drawn below it (this is not the case in the first example though). It should be the result of a message, either from the object itself, or another.
A message sent from outside the diagram can be represented by a message originating from a filled-in circle ("found message" in UML) or from a border of sequence diagram ("gate" in UML).
UML 2 has introduced significant improvements to the capabilities of sequence diagrams [4]. Most of these improvements are based on the idea of interaction fragments which represent smaller pieces of an enclosing interaction. Multiple interaction fragments are combined to create a variety of combined fragments, which are then used to model interactions that include parallelism, conditional branches, optional interactions etc.
Some systems have simple dynamic behavior that can be expressed in terms of specific sequences of messages between a small, fixed number of objects or processes. In such cases sequence diagrams can completely specify the system's behavior. Often, behavior is more complex, e.g. when the set of communicating objects is large or highly variable, when there are many branch points (e.g. exceptions), when

there are complex iterations, or synchronization issues such as resource contention [5]. In such cases, sequence diagrams cannot completely describe the system's behavior, but they can specify typical use cases for the system, small details in its behavior, and simplified overviews of its behavior.

PRESENTATION OF SEQUENCE DIAGRAMS UTILIZED TO IMPLEMENTATION OF A DYNAMICAL SOFTWARE FOR DOING GEOMETRICAL CONSTRUCTIONS

In the achievemnt of the interactive informatics system designed for studying geometry were aimed the following purposes:

- *presenting of theoretical concepts and main results;*
- *interactive presentation of applications for each required subdomain;*
- *achievement of accurate drawings by replacing the pencil and ruler with the mouse.*

By representing the diagrams related to the three steps: analysis, designing and implementation, the interactive informatics system will be described in a clear and concise manner. Utilization of the UML modelling language in the diagrams' achievement is featured by a rich syntactic and semantic rigour, and support for visual modeling.

The sequence diagram is used primarily to show the interactions between objects in the sequential order that those interactions occur. Much like the class diagram, developers typically think sequence diagrams were meant exclusively for them..

The diagram illustrates in figure 1 shows the interactions between objects, which have as purpose the drawing of a parabola. One can notice that there are interactions between nine objects, out of which the objects of Vector<Element2D>, Desen2D and Graphics2D type are already created, and the objects of Element2D, Punct2D, Dreapta2D, MouseEvent and Parabola2D type will instantiate during the interactions.

The diagram illustrates in figure 2 shows the interactions between objects, which have as purpose the drawing of a hyperbola. One can notice that there are interactions between eleven objects, out of which the objects of Vector<Element2D>, Desen2D, Vector<Punct2D> and Graphics2D type are already created, and the objects of Element2D, Parametru, Punct2D, MouseEvent and Hyperbola2D type will instantiate during the interactions.

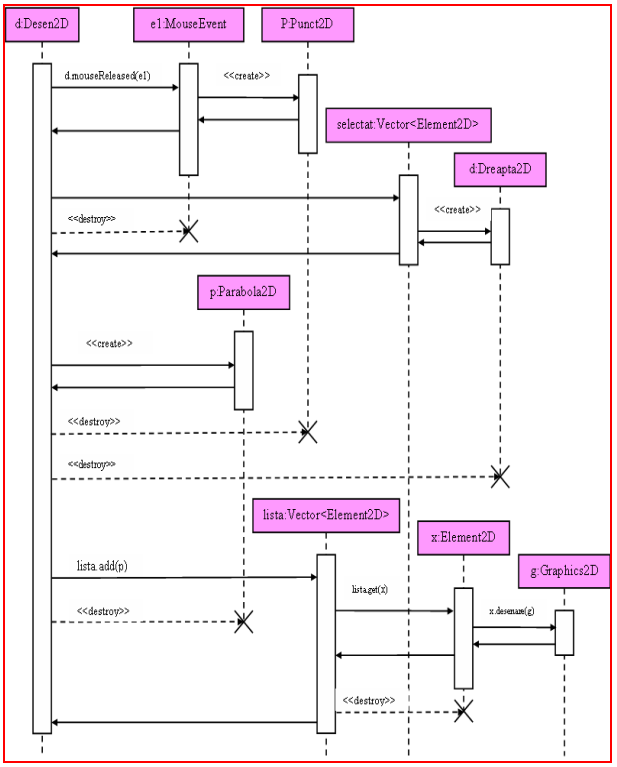

Figure 1. Sequence diagram for drawing a parabola

The diagram illustrates in figure 3 shows the interactions between objects, which have as purpose the drawing the normal to a hyperbola. One can notice that there are interactions between nine objects, out of which the objects of Vector<Element2D>, Desen2D and Graphics2D type are already created, and the objects of Element2D, Punct2D, Dreapta2D, MouseEvent and Hiperbola2D type will instantiate during the interactions.

These objects are represented on Ox axis and, on Oy axis, are represented the messages ordered increasingly in time. At the beginning, the execution's control is undertaken by the object of Desen2D type which creates an instance of the MouseEvent class.

Now, the control is undertaken by this newly created instance that will allow to determining a point. Giving the control to the object of Hiperbola2D type, will verified if the created point belong to the hyperbola.

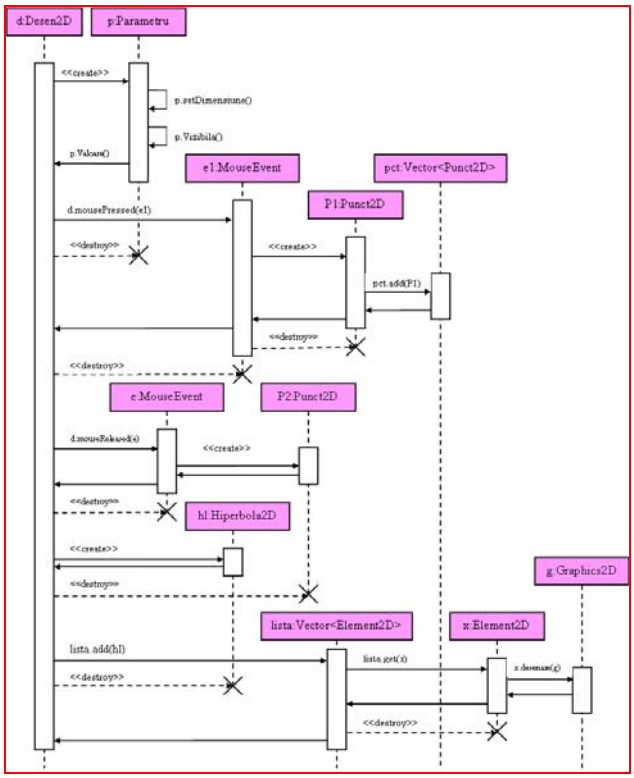

Figure 2. Sequence diagram for drawing a hyperbola

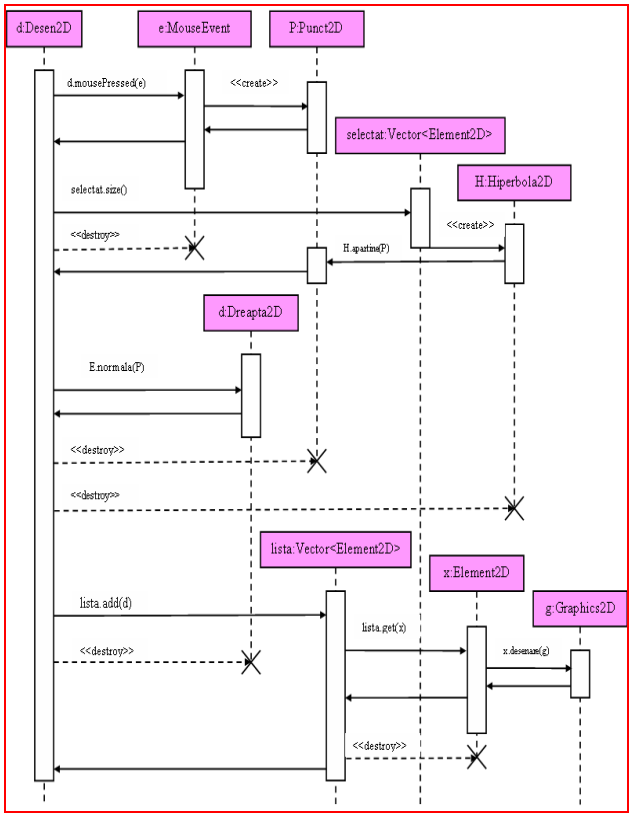

Figure 3. Sequence diagram for drawing a the normal to a hyperbola

Giving back the control to the object of Desen2D type, further will be instantiated the object of Dreapta2D type, representing the normal to hyperbola, then will be destroyed the object of Punct2D type and the object of Hiperbola2D type.

Further, the execution's control is transmitted to the object of Vector<Element2D> type, in order to add the normal previously created in the list of 2D elements of the geometric construction, and then will be destroyed the instance of the Dreapta2D class. Finally, will be redrawn the geometric construction, which will include now also the normal to the previously created hyperbola by using the object of Graphics2D type.

CONCLUSION

The diagrams were achieved by an approach in a new manner, multidisciplinary, of the informatics application, including both the modern pedagogy methods, and the components specific to the discipline to be studied.

Thus, was achieved the connection between the didactic actions and the purposes and objectives scientifically established, by elaborating of new methods and assimilating of new means, capable to increase the school efficiency, allowing the pupils and students to acquire the system required by knowledge's and their application techniques in conditions as optimal possible.

REFERENCES

[1.] Booch G., Rumbaugh J., Jacobson I., The Unified Modeling Language User Guide, Addison Wesley, 1999

[2.] Fowler M., Scott K., UML Distilled: A Brief Guide to the Standard Object Modeling Language, Addison Wesley, Readings MA, USA, 2000

[3.] Odell J., Advanced Object Oriented Analysis & Design using UML, Cambrige University Press, 1998

[4.] Oestereich B., Developing Software with UML, Addison Wesley, 1999

[5.] Rumbaugh J, Jacobson I., Booch G., The Unified Modeling Language Reference Manual, Addison Wesley, 1999

AUTHORS & AFFILIATION

[1.] ANCA IORDAN, [2.] MANUELA PĂNOIU

[1. 2. 3] FACULTY OF ENGINEERING HUNEDOARA, UNIVERSITY "POLITEHNICA" TIMISOARA, ROMANIA