Object Oriented Development of an Interactive Software for Studying of Circle using UML Diagrams

ANCA IORDAN, MANUELA PANOIU, IONEL MUSCALAGIU, RALUCA ROB
Technical University of Timişoara, Engineering Faculty of Hunedoara,
Revoluției 5, 331128 Hunedoara ROMANIA
anca.iordan@fih.upt.ro

Abstract: - This paper presents the necessary steps required for Object Oriented Programming of a computer system used in the study of circle. The modelling of the system is achieved through specific UML diagrams representing the stages of analysis, design and implementation, the system thus being described in a clear and concise manner. The software is very useful to both students and teachers because the mathematics, especially geometry, is difficult to understand for most students.

Key-Words: - Java, UML, Interactive Software, Euclidian Geometry.

1 Introduction

The multimedia technologies transformed the computer into a valuable interlocutor and allowed the students, without going out of the class, to assist the lessons of different emeriti scientists and professors, to communicate with persons located in different countries, to have access to different information [1]. By a single click of the mouse, the student can visit an artistic gallery, read the originals for writing a history paper or visualize information for a narrow profile, which couldn't be found five-ten years ago [2].

One of the main aspects of using computer for lessons is the development of the student's creative thinking. An optimal mean in this case is the introduction in the computational training means of the interactivity elements [3]. The „interactivity" term means „to interact, to influence one-to-another". This property of the computational technologies is absolutely unique compared with television, lectures, books, instructive movies etc.

2 Development stages of interactive software

2.1 Analysis stage

Using UML modelling language, computer system analysis consists in making use case diagram and activity diagrams. To achieve diagrams was used the ArgoUML software [4].

The computer system is described in a clear and concise manner as representing the use cases [5]. Each case describes the interactions between user and system. Use case diagram representation is shown in figure 1. Diagram presented defines the system domain, allowing visualization of the size and sphere of the action for the entire development process. This includes:

- an actor - the user who is external entity with which the system interacts;
- five use cases that describe the functionality of the system;
- relationships between users and use cases (association relationships), and relationships between use cases (dependency and generalization relationships).

For each use case presented in the previous diagram is built an activity diagram. Each diagram shall specify the processes or algorithms that are behind use case analysis.

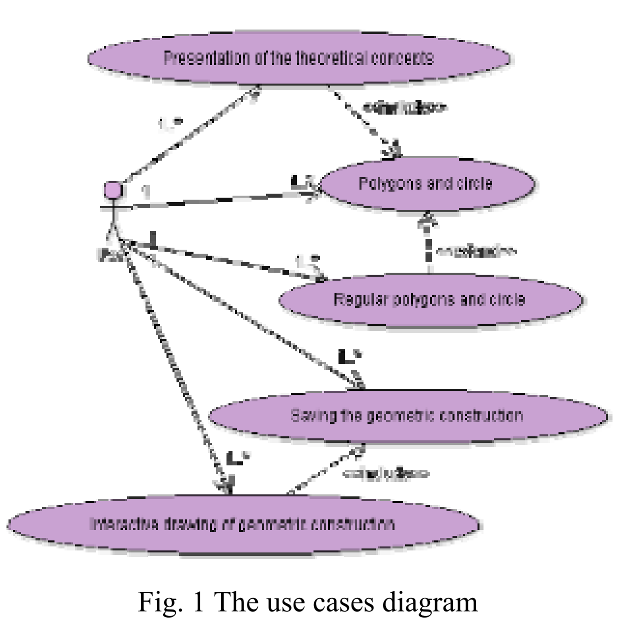
Presentation of the theoretical concepts
Polygons and circle
Regular polygons and circle
Saving the geometric construction
Interactive drawing of geometric construction

Fig. 1 The use cases diagram

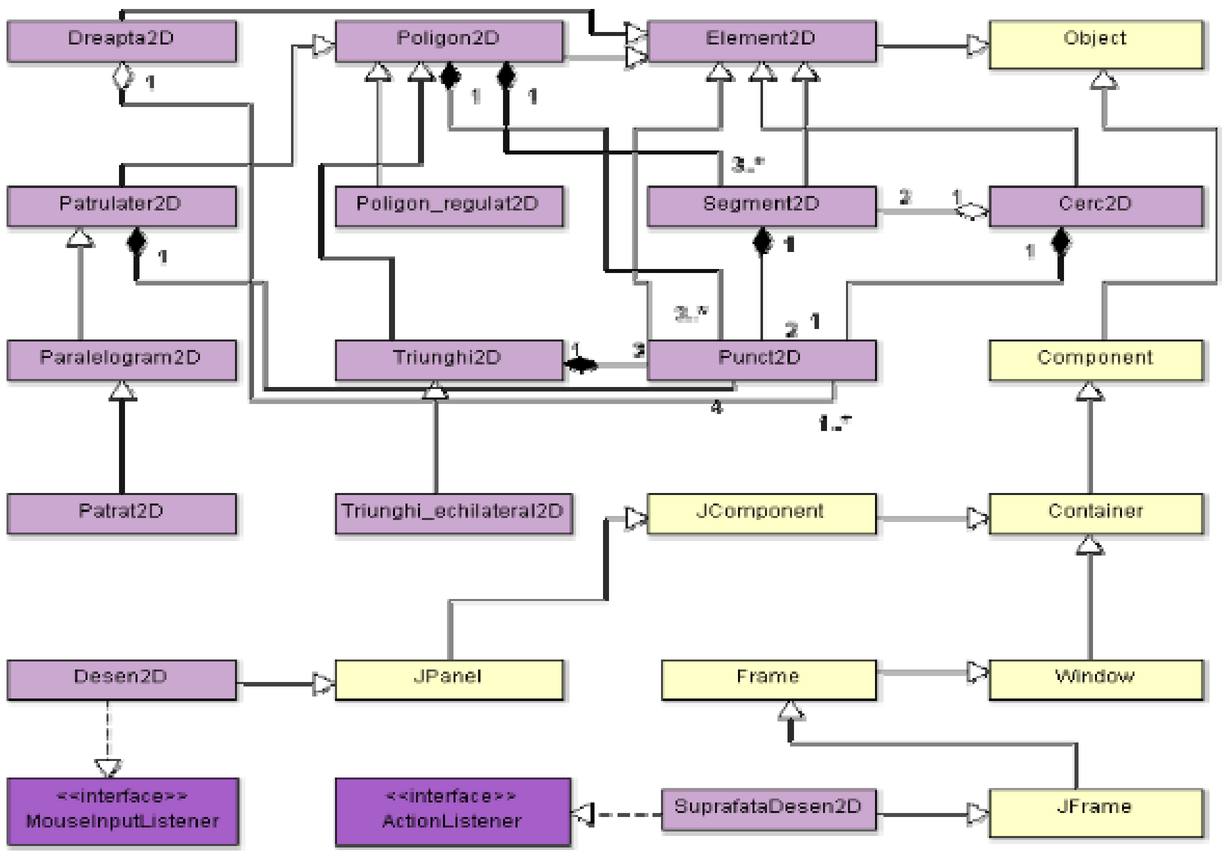

Fig. 2 Class diagram

2.2 Design stage

2.2.1 Class diagram

Conceptual modelling allows identifying the most important concepts for the computer system [6]. Inheritance was not used only as a generalization mechanism, which is when derived classes are specializations of the base class.

In figure 2 are presented inheritance relationships, realization relationships, composition relationships and aggregation relationships. We can observe that the *SuprafataDesen2D* class inherit attributes and methods of the *JFrame* class, but implements the *ActionListener* interface. *Desen2D* class inherit attributes and methods of the *JPanel* class, but implements *MouseInputListener* interface.

Between instances of the classes presented in figure 2 there are especially composition and aggregation relationships. In the composition relationship, unlike the aggregation relationship, the instance can not exist without the party objects. Analyzing figure we can observe that an instance of *Segment2D* type consists in two objects of *Punct2D* type. Aggregation relationship is an association where it's specified who is integer and who is a party. For example, an object of *Segment2D* type represents a part from an object of *Cerc2D* type.

2.2.2 Sequence diagrams

Description of behaviour involves two aspects: structural description of participants and description of models of communication. The communication model of the instance witch play one role to fulfill a specific purpose is called interaction.

The purpose of interaction diagram is to specify how to carry out an operation or a use case [7], modelling the behaviour of a set of objects in a certain context. Interaction context may be the system (subsystem), or class operation.

Objects can be concrete things, or prototypes, among them setting the semantic connections. There are two forms of interaction diagrams based on the same basic information, but each focuses on another aspect of interaction: sequence diagrams and collaboration diagrams.

Sequence diagram emphasizes the temporal aspect, being suitable for real-time specifications and complex scenarios [8]. These diagrams determine the objects and classes involved in a scenario and sequence of messages sent between objects necessary to execute script functionality. Sequence diagrams are associated with a use case.

Diagram presented in figure 3 renders the interactions between objects that are designed to drawing the circumcircle of a regular polygon and the centre of this circle.

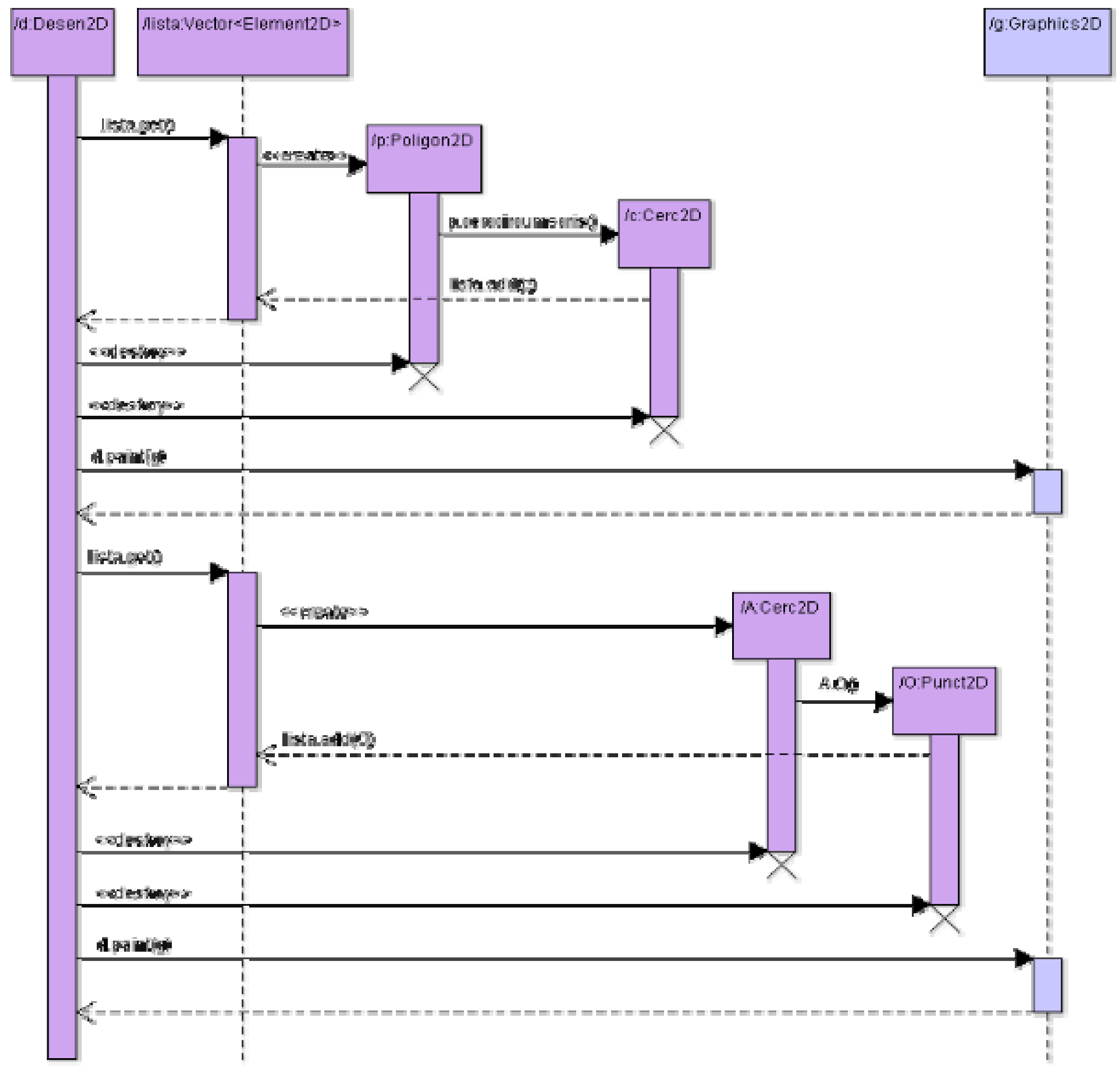

Fig. 3 Sequence diagram for drawing the circumcircle of a regular polygon

We can observe that there are interactions between the 7 objects, of which the objects of *Desen2D* type, Vector<Element2D> type and *Graphics2D* type are already created and the objects of *Poligon2D* type, *Cerc2D* type and *Punct2D* type will instantiate during interactions.

At first the execution control is taken by the object of *Desen2D* type. Following an event that interacts with the *Desen2D* object is transmitted the control of the object of *Vector<Element2D>* type that creates an instance of *Polygon2D* class. Following is created an instance of *Cerc2D* class. Control is transmitted to the object of *Vector<Element2D>* type to add the object previously created. The control will be given to the object of *Desen2D* type that will destroy the object of *Polygon2D* type and the object of *Cerc2D* type. Through interaction with *Graphics2D* object will redraw the circumcircle of a regular polygon. We can observe that lifeline of the *Cerc2D* object is interrupt, by marking an X, the message appears bearing the stereotype <<*destroy*>>.

Following an event that interacts with the *Desen2D* object is transmitted the control of the object of *Vector<Element2D>* type that creates an instance of *Cerc2D* class. Following is created an instance of *Punct2D* class. Control is transmitted to the object of *Vector<Element2D>* type to add the object previously created.

The control will be given to the object of *Desen2D* type that will destroy the object of *Cerc2D* type and the object of *Punct2D* type. Through interaction with *Graphics2D* object will redraw the centre of circumcircle of a regular polygon.

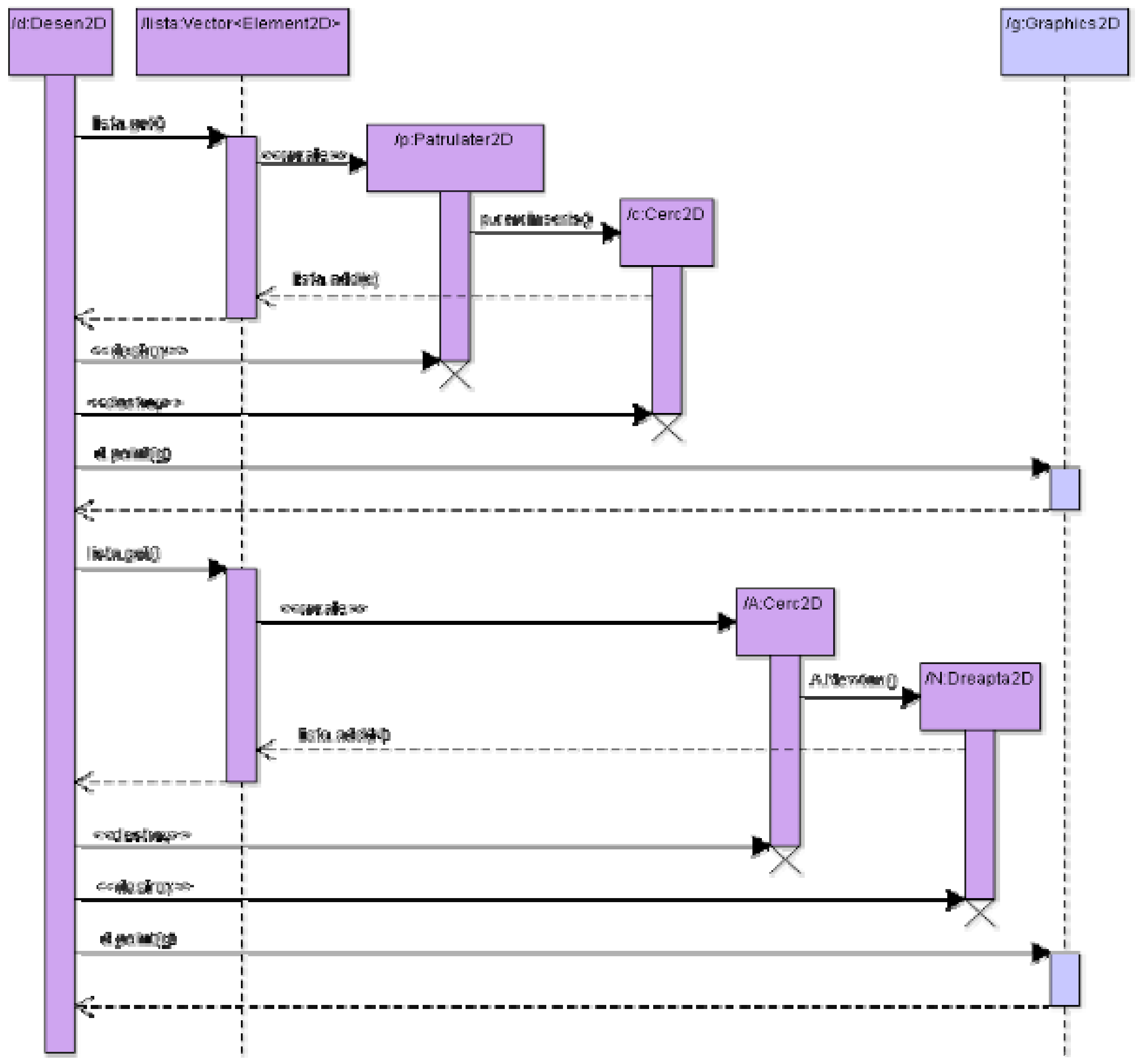

Fig. 4 Sequence diagram for drawing the incircle of a circumscribed quadrilateral and the Newton line

Diagram presented in figure 4 renders the interactions between objects that are designed to drawing the incircle of a circumscribed quadrilaterals and the Newton line.

We can observe that there are interactions between the 7 objects, of which the objects of *Desen2D* type, Vector<Element2D> type and *Graphics2D* type are already created and the objects of *Patrulater2D* type, *Cerc2D* type and *Dreapta2D* type will instantiate during interactions. At first the execution control is taken by the object of *Desen2D* type. Following an event that interacts with the *Desen2D* object is transmitted the control of the object of *Vector<Element2D>* type that creates an instance of *Patrulater2D* class. Following is created an instance of *Cerc2D* class. Control is transmitted to the object of *Vector<Element2D>* type to add the object previously created.

The control will be given to the object of *Desen2D* type that will destroy the object of *Patrulater2D* type and the object of *Cerc2D* type. Through interaction with *Graphics2D* object will redraw the incircle of a circumscribed quadrilateral. We can observe that lifeline of the *Cerc2D* object is interrupt, by marking an X, the message appears bearing the stereotype <<*destroy*>>.

Following an event that interacts with the *Desen2D* object is transmitted the control of the object of *Vector<Element2D>* type that creates an instance of *Cerc2D* class. Following is created an instance of *Dreapta2D* class. Control is transmitted to the object of *Vector<Element2D>* type to add the object previously created.

The control will be given to the object of *Desen2D* type that will destroy the object of *Cerc2D* type and the object of *Dreapta2D* type. Through interaction with *Graphics2D* object will redraw the Newton line of a circumscribed quadrilateral.

2.2.3 Collaboration diagrams

Collaboration diagrams describe the behaviour of a set of objects in a certain context with an emphasis on organizing the objects involved in the interaction [9]. These diagrams are graphs witch has in peaks quality, objects that participate to the interaction, and the arcs represent links between instances. Diagram presented in figure 5 renders the interactions between objects that allow drawing the incircle of a circumscribed quadrilateral and the Newton line.

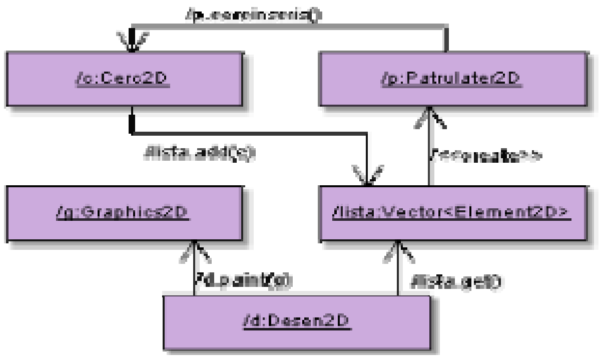

Fig. 5 Collaboration diagram for drawing the incircle of a circumscribed quadrilateral

2.3 Implementation stage

Component diagram is similar to packages diagram, allowing visualization of how the system is divided and the dependencies between modules [10]. Component diagram put emphasis on software physical elements and not on the logical elements like in case of packages.

The diagram in figure 6 describes the collection of components that together provide system functionality.

Central component of the diagram is *SuprafataDesen2D.class,* a component obtained by transforming by the Java compiler into executable code of the *SuprafataDesen2D.java* component. As can be seen that component interacts directly with components *Desen2D.class*. This component interacts with *Element2D.class* component.

3 Graphical user interface

The application was implemented in Java as independent application [11]. The interactive system allows to drawing circles in diverse modes, but presents both theoretical results and certain types of solved problems. Among the most important operations we mention:

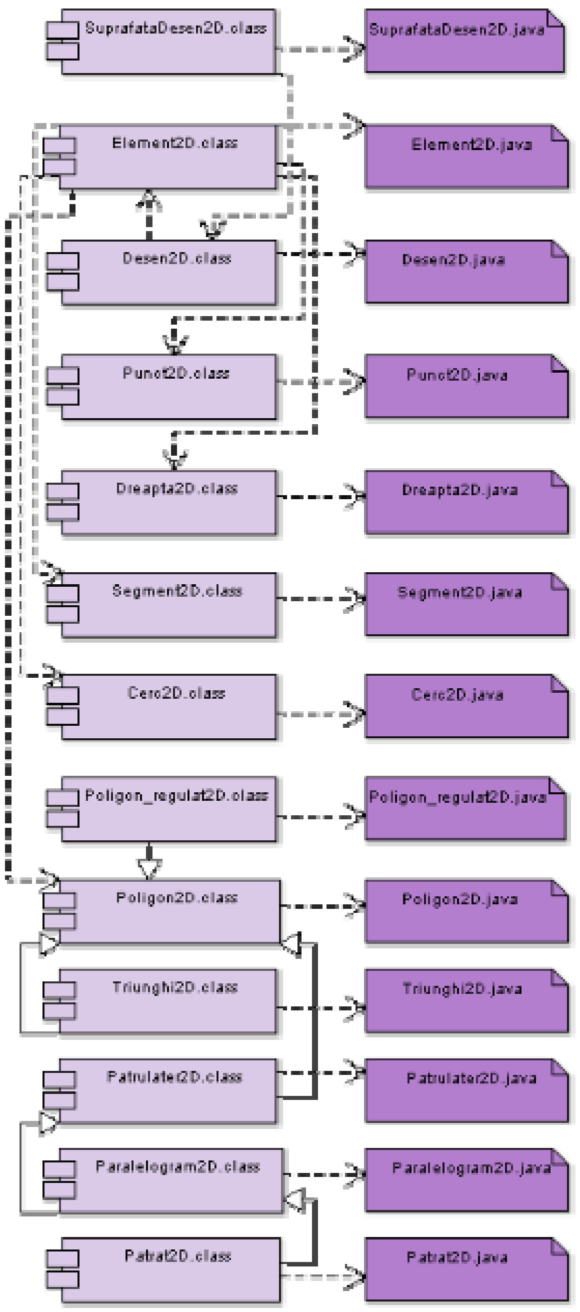

Fig. 6 Component diagram

- circles determined by three point;
- circles when is specified the centre and radius;
- circles which fulfill certain conditions:

- the incircle and the three excircles for a given triangle;
- the circumcircle of a given triangle;
- the Euler circle of a given triangle;
- the Lemoine circles for a given triangle;
- the Taylor circle of a given triangle;
- the incircle of a circumscribed quadrilateral (figure 7);

- the circumcircle of a cyclic quadrilateral (figure 8);
- the circumcircle of a regular polygon.

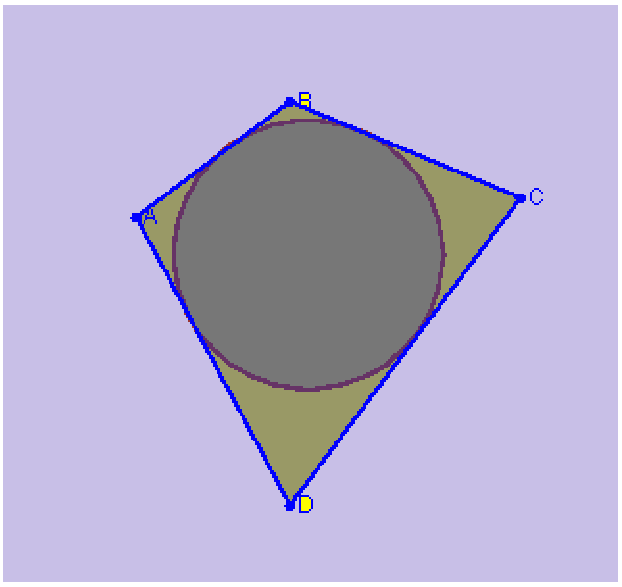

Fig. 7 The incircle of a circumscribed quadrilateral

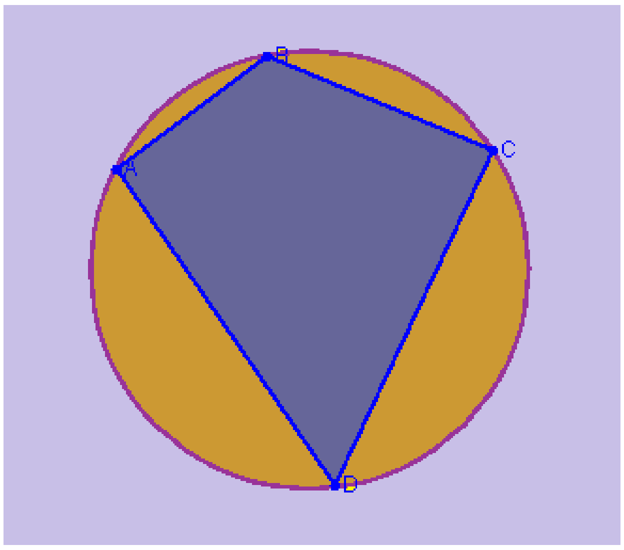

Fig. 8 The circumcircle of a cyclic quadrilateral

4 Conclusion

Through the diagram representation all three phases: analysis, design and implementation, the educational informatics system has been described in a clear and concise manner. The use of the UML modelling language for the creation of the diagrams is characterized by rigorous syntactic, rich semantic and visual modelling support.

The diagrams were made using a new approach, multidisciplinary of the informatics application, encompassing both modern pedagogy methods and discipline-specific components. The link between teaching activities and scientific goals and objectives was established through the development of the new methods and the assimilation of new ways, capable of enhancing school performance, enabling students to acquire the knowledge and techniques required and apply them in optimum conditions.

References:

[1] C. Cucos, *Pedagogy*, Polirom Press, Iasi, 2002

[2] M. Panoiu, I. Muscalagiu, C. Panoiu, M. Raich, Educational Software for the Study the Performances of Some Known Parallel and Sequential Algorithms, *WSEAS Transactions on Information Science and Applications*, Vol. 7, 2010, pp. 1271-1283

[3] T. H. Wang, K. H. Wang, S. C. Huang, Designing a Web-based assessment environment for improving pre-service teacher assessment literacy, *Computers & Education, 2008,* pp. 448-462

[4] http://argouml.tigris.org

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

[6] J. Odell, “Advanced Object Oriented Analysis& Design using UML”, *Cambrige University Press*, 1998

[7] J. Rumbaugh, I. Jacobson, G. Booch, “The Unified Modeling Language Reference Manual”, *Addison Wesley*, 1999

[8] S. Bennet, S. McRobb, R. Farmer, “Object Oriented Systems Analysis and Design”, *McGraw Hill*, 1999

[9] G. Booch, J. Rumbaugh, I. Jacobson, “The Unified Modeling Language User Guide”, *Addison Wesley*, 1999

[10] J. Cheesman, J. Daniels, UML Components: A Simple Process for Specifying Component-Based Software, *Addison- Wesley*, Mass, USA, 2000

[11] S. Tănasă, C. Olaru, S. Andrei, “Java de la 0 la expert”, *Polirom Press*, Iasi, 2007