Using UML Diagrams for Object Oriented Implementation of an Interactive Software for Studying the Circle

A. Iordan, M. Panoiu, I. Muscalagiu, R. Rob

***Abstract*—** This paper presents the necessary steps required for Object Oriented Implementation of a computer system used in the study of circle. The modeling 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.

***Keywords*—**Java, UML, Interactive Software, Euclidian Geometry.

I. 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.

A. Iordan is with the Engineering Faculty of Hunedoara, Polytechnic University of Timisoara, 331128 Hunedoara, ROMANIA (corresponding author to provide phone: +040-254-207523; e-mail: anca.iordan@fih.upt.ro).

M. Panoiu is with the Engineering Faculty of Hunedoara, Polytechnic University of Timisoara, 331128 Hunedoara, ROMANIA (corresponding author to provide phone: +040-254-207537; e-mail: manuela.panoiu@fih.upt.ro).

I. Muscalagiu is with the Engineering Faculty of Hunedoara, Polytechnic University of Timisoara, 331128 Hunedoara, ROMANIA (corresponding author to provide phone: +040-254-207537; e-mail: ionel.muscalagiu@fih.upt.ro).

R. Rob is with the Engineering Faculty of Hunedoara, Polytechnic University of Timisoara, 331128 Hunedoara, ROMANIA (corresponding author to provide phone: +040-254-207523; e-mail: raluca.rob@fih.upt.ro).

II. DEVELOPMENT STAGES OF INTERACTIVE SOFTWARE

A. Analysis stage

Using UML modeling 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.

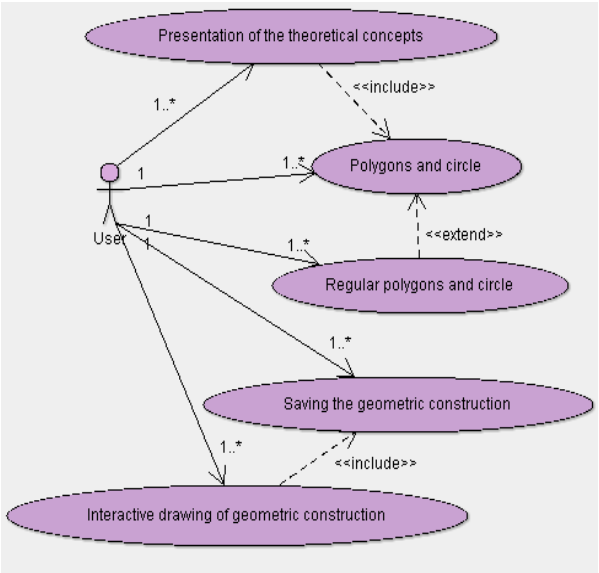

Fig. 1. The use cases diagram

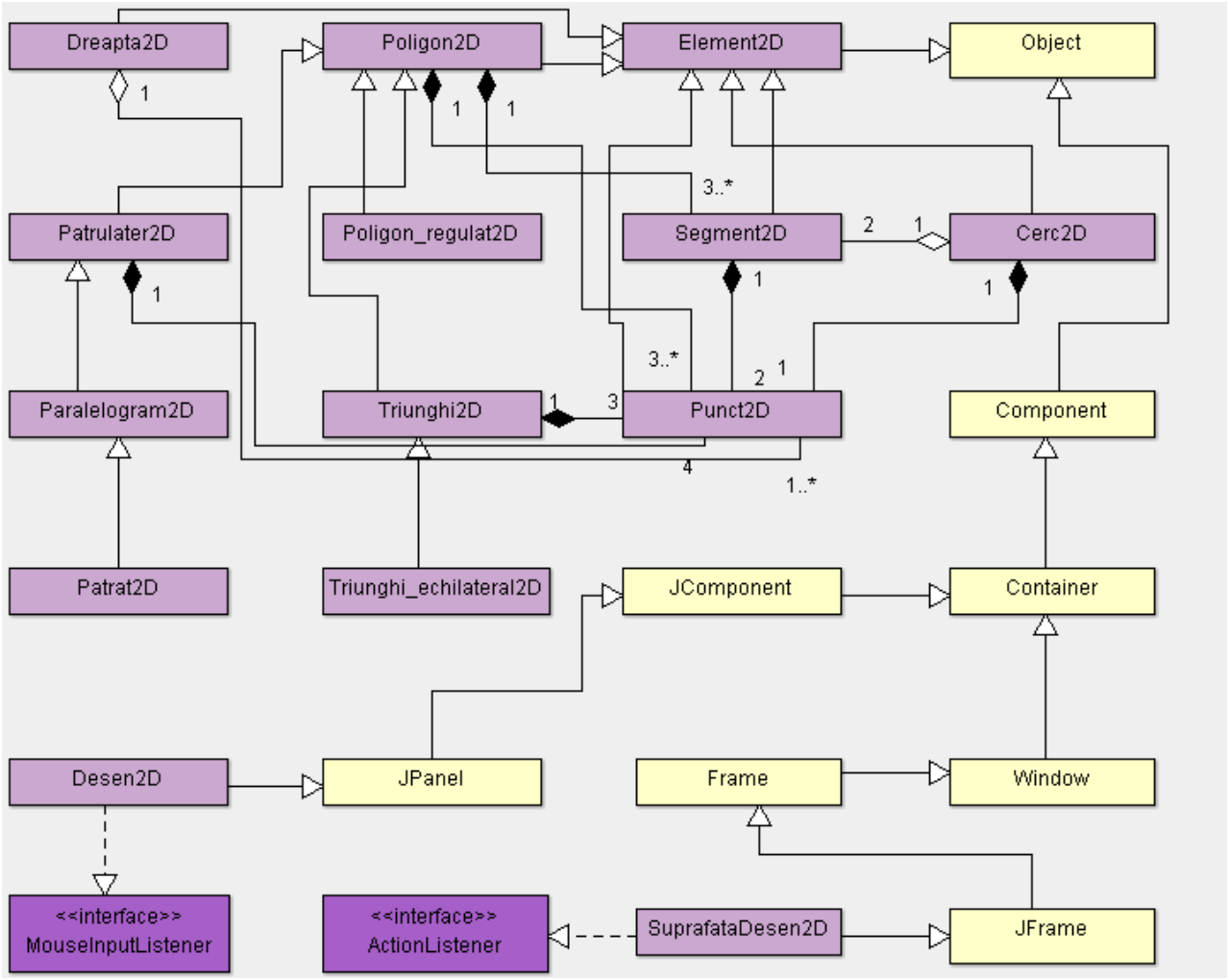

Fig. 2. Class diagram

B. Design stage – Class diagram

Conceptual modeling 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.

C. Design stage – Sequence diagram

Description of behavior 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], modeling the behavior 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.

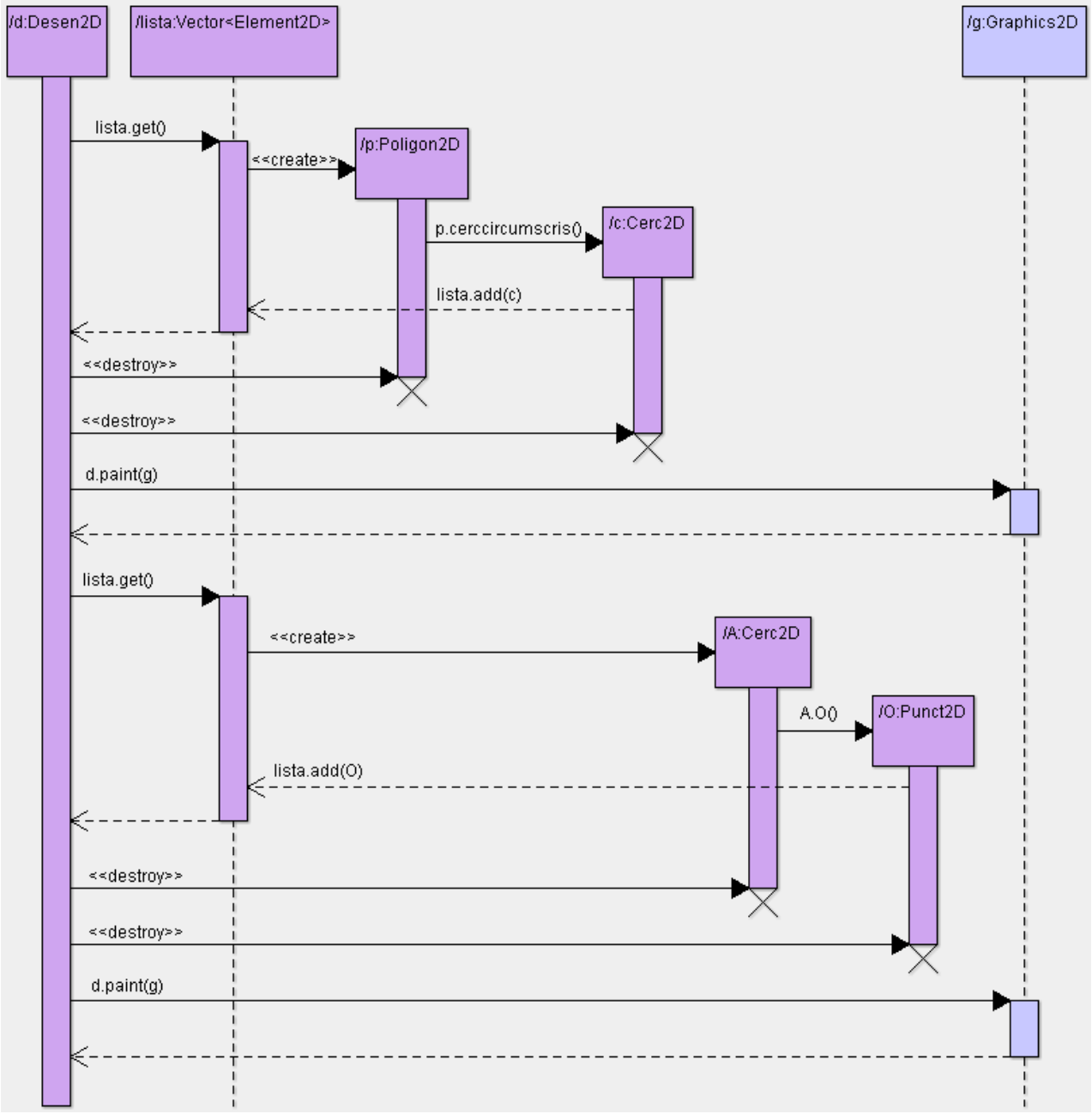

Fig. 3 Sequence diagram for drawing the circumcircle of a regular polygon

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.

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.

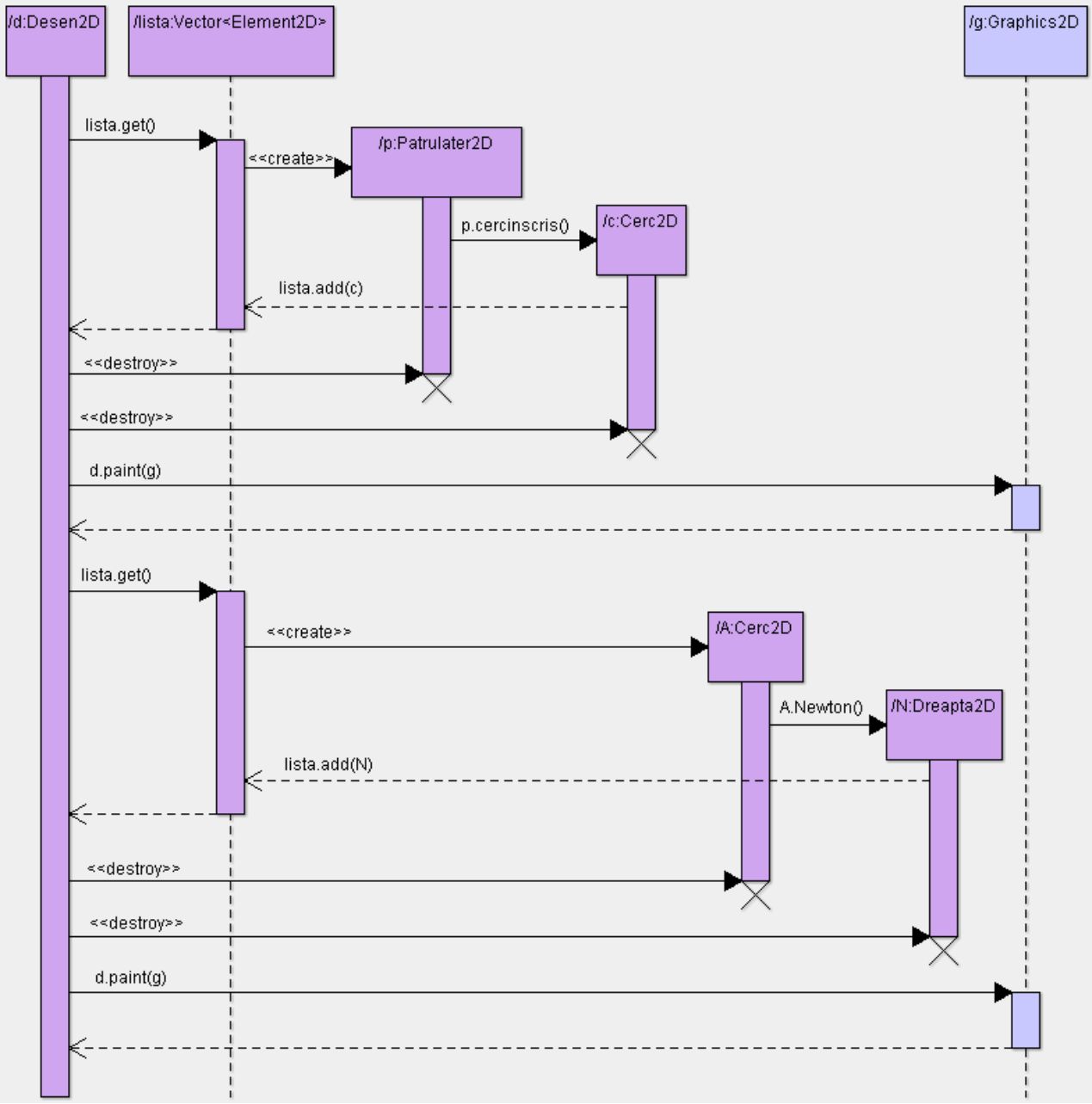

Fig. 4 Sequence diagram for drawing the incircle of a circumscribed quadrilateral and the Newton line

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.

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.

D. Design stage – Sequence diagram

Collaboration diagrams describe the behavior 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.

Diagram presented in figure 6 renders the interactions between objects that allow drawing the circumcircle of a regular polygon.

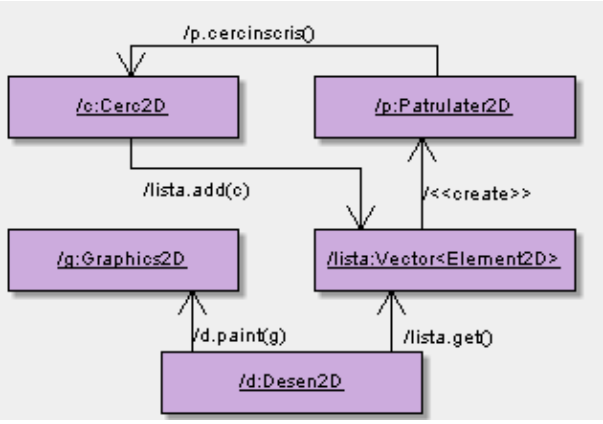

Fig. 5 Collaboration diagram for drawing the incircle of a circumscribed quadrilateral

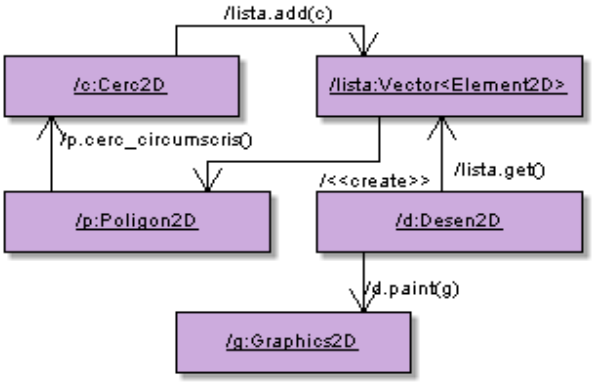

Fig. 6 Collaboration diagram for drawing the circumcircle of a regular polygon

E. 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 7 describes the collection of components that together provide system functionality.

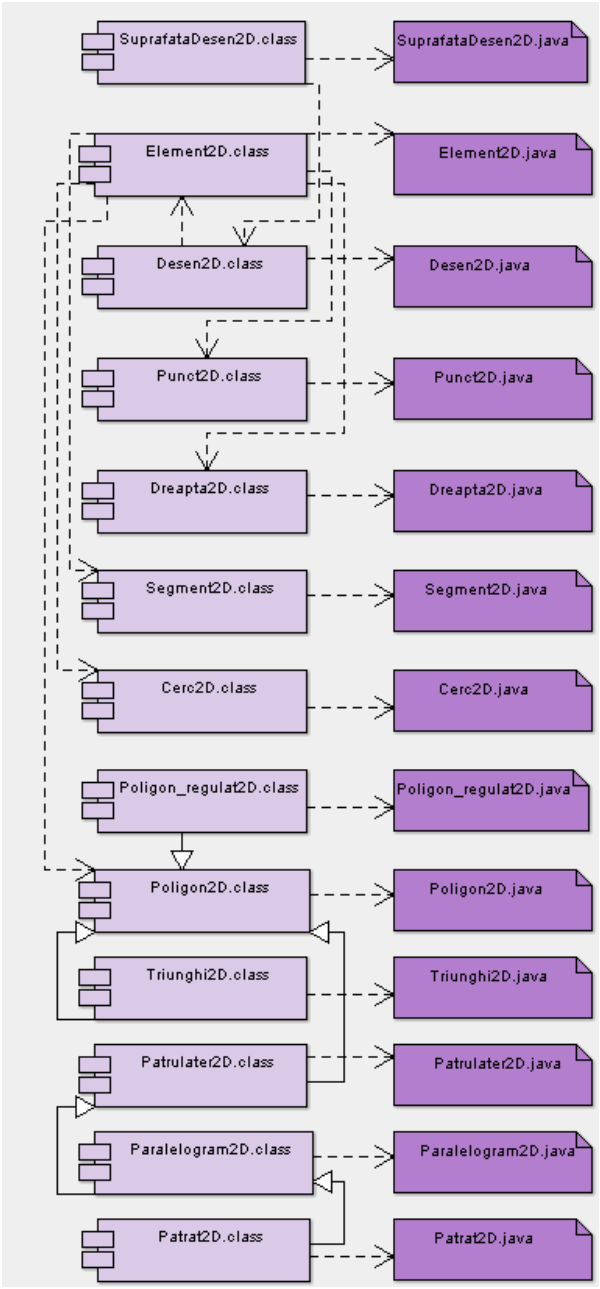

Fig. 7 Component diagram

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, which is obtained by transforming by the Java compiler into executable code of the *Element2D 2D.java* component.

III. 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:

- drawing-up of circles determined by three point;
- drawing-up of circles when is specified the centre and radius;
- drawing-up of circles which fulfill certain conditions:
  - the incircle and the three excircles for a given triangle (figure 8);
  - the circumcircle of a given triangle;
  - the Euler circle of a given triangle (figure 9);
  - the Lemoine circles for a given triangle (figure 10);
  - the Taylor circle of a given triangle (figure 11);
  - the incircle of a circumscribed quadrilateral (figure 12);
  - the circumcircle of a cyclic quadrilateral (figure 13);
- the circumcircle of a regular polygon.

The implementation of the method which obtains the incircle for a given triangle is presented forwards:

```
public Cerc2D cerc_inscris() {
        double p=(sg[0].getLungime()+
sg[1].getLungime()+sg[2].getLungime())/2;
        double r=arie()/p;
        return new Cerc2D(I(),r);
    }
```

The implementation of the methods which obtains the centre of the incircle for a given triangle is presented forwards:

```
public Punct2D I() {
        Dreapta2D d1=new Dreapta2D(V[0],
picior_bisectoare(V[0]));
        Dreapta2D d2=new Dreapta2D(V[1],
picior_bisectoare(V[1]));
        Punct2D i=new Punct2D(d1.intersectie(d2));
        return i;
}
public Punct2D picior_bisectoare(Punct2D p) {
        int i=0;
        if (p.coincid(V[0])) i=1;
        if (p.coincid(V[1])) i=2;
        if (p.coincid(V[2])) i=0;
        double x,y,k;
        if (i==1) {
            k=V[0].distanta(V[1])/V[0].distanta(V[2]);
            x=(V[1].getX()+k*V[2].getX())/(1+k);
            y=(V[1].getY()+k*V[2].getY())/(1+k);
        }
        else if (i==2) {
            k=V[1].distanta(V[2])/V[1].distanta(V[0]);
            x=(V[2].getX()+k*V[0].getX())/(1+k);
            y=(V[2].getY()+k*V[0].getY())/(1+k);
        }
        else {
            k=V[2].distanta(V[0])/V[2].distanta(V[1]);
            x=(V[0].getX()+k*V[1].getX())/(1+k);
            y=(V[0].getY()+k*V[1].getY())/(1+k);
        }
        return new Punct2D(x,y);
}
public Punct2D intersectie(Dreapta2D d){
        Punct2D p=new Punct2D();
        double dx,dy;
        dx=-C*d.B+d.C*B;
        dy=A*d.B-d.A*B;
        p.setX(dx/dy);
        if (ind==3) p.setY(d.m*p.getX()+d.n);
         else p.setY(m*p.getX()+n);
         return p;
}
```

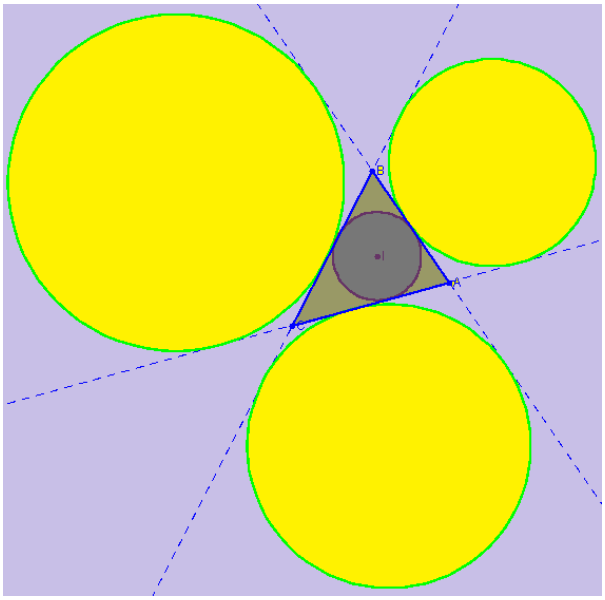

Fig. 8 The incircle and the three excircles for a given triangle

The implementation of the methods which obtains the excircle for a given triangle is presented forwards:

```
public Cerc2D cerc_exinscris(Punct2D p) {
    int i;
    if (p.apartine(V[0])) i=0;
    else if (p.apartine(V[1])) i=1;
    else i=2;
    Dreapta2D d1,d2,d3;
    if (i==0) {
    d1=new Dreapta2D(bisectoare_int(V[0]).getSuport());
    d2=new Dreapta2D(bisectoare_int(V[1]). getSuport());
    d3=new Dreapta2D(d2. perpendiculara(V[1]));
    }
    else if (i==1) {
    d1=new Dreapta2D(bisectoare_int(V[1]).getSuport());
    d2=new Dreapta2D(bisectoare_int(V[0]).getSuport());
    d3=new Dreapta2D(d2.perpendiculara(V[0]));
    }
    else {
    d1=new Dreapta2D(bisectoare_int(V[2]). getSuport());
    d2=new Dreapta2D(bisectoare_int(V[1]).getSuport());
    d3=new Dreapta2D(d2.perpendiculara(V[1]));
  }
```

```
        Punct2D I=new Punct2D(d3.intersectie(d1));
        if (i==0) d1.setare(V[0],V[1]);
        else if (i==1) d1.setare(V[1],V[0]);
        else d1.setare(V[2],V[1]);
        d2.setare(d1.perpendiculara(I));
        Punct2D M=new Punct2D(
        d2.intersectie(d1));
        double r=I.distanta(M);
        return new Cerc2D(I,r);
    }
    public Semidreapta2D bisectoare_int(Punct2D p)
    {
        return new Semidreapta2D(p, picior_bisectoare(p));
    }
    public Dreapta2D perpendiculara(Punct2D p){
        double m1;
        if (m==Double.POSITIVE_INFINITY)
            m1=0;
        else if (m==0)
            m1=Double.POSITIVE_INFINITY;
        else m1=-1/m;
        Dreapta2D d=new Dreapta2D(p,m1);
        return d;
    }
```

The method of the class *Triunghi2D* which obtains the Euler circle (figure 9) for a given triangle is presented forwards:

```
    public Cerc2D cerc_Euler() {
         double r=sg[0].getLungime()*sg[1].getLungime()*
sg[2].getLungime();
         r=r/(8*arie());
         return new Cerc2D(W(),r);
    }
```

The methods which obtains the centre of the Euler circle for a given triangle is presented forwards:

```
    public Punct2D W() {
        Segment2D s=new Segment2D(O(),H());
        return s.mijloc();
   }
    public Punct2D O() {
        Dreapta2D d1=new Dreapta2D(sg[0]. mediatoare());
        Dreapta2D d2=new Dreapta2D(sg[1].mediatoare());
        Punct2D o=new Punct2D(d1.intersectie(d2));
        return o;
    }
    public Punct2D H() {
        return new Punct2D(inaltime(V[0]).intersectie
(inaltime(V[1])));
    }
    public Punct2D mijloc() {
        double x,y;
        x=(A1.getX()+A2.getX())/2;
        y=(A1.getY()+A2.getY())/2;
        Punct2D M=new Punct2D(x,y);
        return M;
    }
    public Dreapta2D mediatoare() {
        Punct2D M=new Punct2D(mijloc());
        Dreapta2D med=new Dreapta2D(d.perpendiculara(M));
        return med;
    }
    public Dreapta2D inaltime(Punct2D p) {
        int i=0;
        if (p.coincid(V[0])) i=1;
        if (p.coincid(V[1])) i=2;
        if (p.coincid(V[2])) i=0;
    return new Dreapta2D(sg[i].getSuport().perpendiculara(p));
    }
```

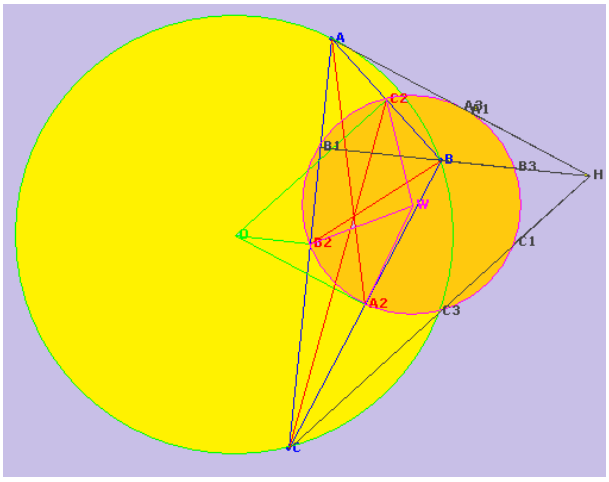

Fig. 9 The Euler circle of a given triangle

The method of the class *Triunghi2D* which obtains the first Lemoine circle (figure 10) for a given triangle is presented forwards:

```
    public Cerc2D cerc_Lemoine() {
        double r=sg[0].getLungime()*sg[1]. getLungime()
*sg[2].getLungime();
        r=r/(8*arie());
        return new Cerc2D(K(),r);
    }
```

The methods of the class *Triunghi2D* which obtains the centre of the first Lemoine circle for a given triangle is presented forwards:

```
    public Punct2D K() {
         return new Punct2D(simediana(V[0]).getSuport().
intersectie(simediana(V[1]).getSuport())); }
    public Segment2D simediana(Punct2D p) {
        return new Segment2D(p, piciorsimediana(p));
    }
    public Punct2D piciorsimediana(Punct2D p)
    {
        int i=0;
        if (p.apartine(V[0])) i=1;
        if (p.apartine(V[1])) i=2;
        if (p.apartine(V[2])) i=0;
        Punct2D M=new Punct2D(sg[i].mijloc());
        double x,y,k;
        if (i==1) {
            k=V[0].distanta(V[1])/V[0].distanta(V[2]);
            x=(V[1].getX()+k*V[2].getX())/(1+k);
            y=(V[1].getY()+k*V[2].getY())/(1+k);
```

```
        }
        else if (i==2) {
             k=V[1].distanta(V[2])/V[1].distanta(V[0]);
            x=(V[2].getX()+k*V[0].getX())/(1+k);
            y=(V[2].getY()+k*V[0].getY())/(1+k);
        }
      else {
            k=V[2].distanta(V[0])/V[2].distanta(V[1]);
            x=(V[0].getX()+k*V[1].getX())/(1+k);
            y=(V[0].getY()+k*V[1].getY())/(1+k);
        }
        Punct2D D=new Punct2D(x,y);
        Simetrie2D s=new Simetrie2D(D);
        return new Punct2D(s.izometrie(M));
}
```

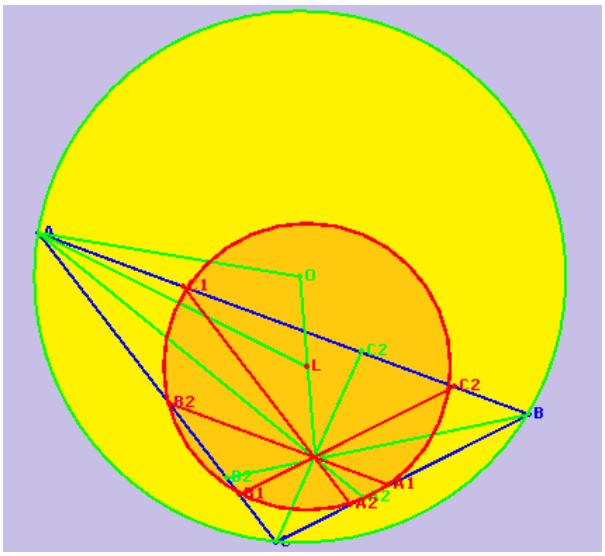

Fig. 10 The Lemoine circles for a given triangle

The implementation of the methods which obtains the Taylor circle (figure 11) for a given triangle is presented forwards:

```
    public Cerc2D cerc_Taylor() {
        Punct2D A1=new Punct2D(picior_inaltime(V[0]));
        Dreapta2D d1=new Dreapta2D((sg[0].
getSuport()).perpendiculara(A1));
        Punct2D A2=new Punct2D(d1.intersectie
(sg[0].getSuport()));
        d1=new Dreapta2D((sg[2].getSuport()).
perpendiculara(A1));
        Punct2D A3=new Punct2D(d1.intersectie
(sg[2]. getSuport()));
        Punct2D B1=new Punct2D( picior_inaltime(V[1]));
        d1=new Dreapta2D((sg[0].getSuport()).
perpendiculara(B1));
        Punct2D B2=new Punct2D(d1.intersectie
(sg[0].getSuport()));
        return new Cerc2D(A2,A3,B2);
    }
```

The methods which obtains the incircle of a circumscribed quadrilateral (figure 12) is presented forwards:

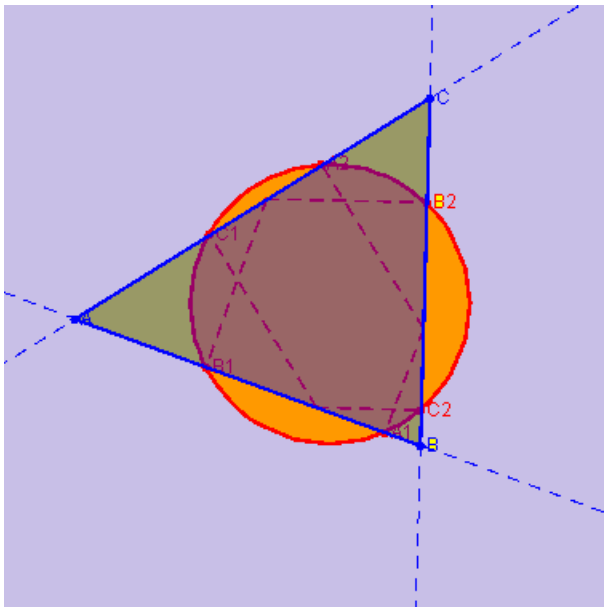

Fig.11 The Taylor circle of a given triangle

```
    public Cerc2D cerc_inscris() {
        Cerc2D C;
        Triunghi2D t;
        if (!sg[0].getSuport().paralele (sg[2].getSuport()))
          {
            t=new Triunghi2D(V[0],V[3],
sg[0].getSuport().intersectie(sg[2].getSuport()));
            C=new Cerc2D(t.cerc_inscris());
          }
        else if (!sg[1].getSuport().paralele(sg[3]. getSuport()))
          {
            t=new Triunghi2D(V[1],V[0],sg[1].getSuport()
.intersectie(sg[3].getSuport()));
            C=new Cerc2D(t.cerc_inscris());
           }
        else
          {
            Punct2D O=new Punct2D(new Dreapta2D(V[0],
V[2]).intersectie(new Dreapta2D(V[1],V[3])));
            double raza=(sg[0].getSuport()).distanta(V[2])/2;
            C=new Cerc2D(O,raza);
           }
        return C;
    }
```

The method of the class *Patrulater2D* which verifying if the quadrilateral is circumscribed is presented forwards:

```
    public boolean circumscriptibil() {
        return sg[0].lungime()+sg[2].lungime()==sg[1].
lungime()+sg[3].lungime();
    }
```

The method of the class *Patrulater2D* which obtains the circumcircle of a cyclic quadrilateral (figure 13) is presented forwards:

```
    public Cerc2D cerc_circumscris() {
         return new Cerc2D(V[0],V[1],V[2]);
    }
```

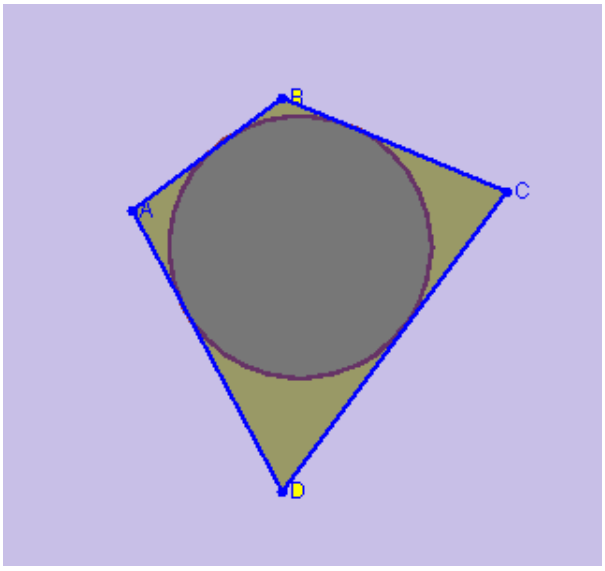

Fig. 12 The incircle of a circumscribed quadrilateral

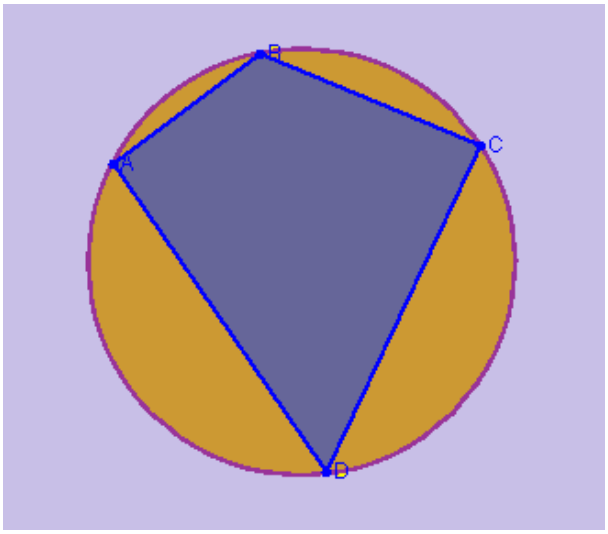

Fig. 13 The circumcircle of a cyclic quadrilateral

The method of the class *Patrulater2D* which verifying if the quadrilateral is cyclic is presented forwards:

```
public boolean inscriptibil() {
    double s=U[0].getMasura()+ U[2].getMasura();
    return Math.floor(s)==180;
}
```

IV. 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 modeling language for the creation of the diagrams is characterized by rigorous syntactic, rich semantic and visual modeling 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] G. Gallitano, K. Jackson, *The Content and Format of a Professional Development Program and Its Attitudinal Effect on Teachers of Mathematics,* INTERNATIONAL JOURNAL OF EDUCATION AND INFORMATION TECHNOLOGIES (Naun), Issue 3, Volume 5, 2011, pp. 336-343

[2] Dimitrios Rigas, Khaled Ayad, *Using edutainment in e-learning application: an empirical study,* INTERNATIONAL JOURNAL OF COMPUTERS (Naun), Issue 1, Volume 4, 2010, pp. 36-43

[3] R. Pereira, I. Brito, G. Machado, T. Malheiro, E. Vaz, M. Flores, J. Figueiredo, P. Pereira, A. Jesus, New e-learning objects for the Mathematics courses from Engineering degrees: Design and Implementation of Question Banks in Maple T.A. using LaTeX, INTERNATIONAL JOURNAL OF EDUCATION AND INFORMATION TECHNOLOGIES (Naun), Issue 1, Volume 4, 2010, pp. 7-14

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

[5] M. Fowler, K. Scott, "*UML Distilled: A Brief Guide to the Standard Object Modelling 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 Modelling 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 Modelling 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*", Polirom Press, Iasi, 2007

A. Iordan, born in 1979, graduated from the Computer Science Faculty, Babes-Bolyai University of Cluj-Napoca in 2002. She received her PhD degree in Computer Science in 2009 at Polytechnic University of Timisoara and is currently assistant at the Electrical Engineering and Industrial Informatics Department of Engineering Faculty of Hunedoara, Polytechnic University of Timisoara, Romania. Her research interests include Data Structures and Algorithms, Software Engineering and Educational Software. She has until now published over 50 research papers in journals and conferences.

M. Panoiu, born in 1965, graduated from the Computer Science Faculty, Polytechnic University of Timisoara in 1989. She received her PhD degree in Electrical Engineering in 2001 and is currently Assistant Professor at the Electrical Engineering and Industrial Informatics Department of Engineering Faculty of Hunedoara, Polytechnic University of Timisoara, Romania. Her research interests focus on advanced computer programming, modeling and simulating systems, and artificial intelligence. She has until now published over 80 research papers in journals and conferences and participated in 9 research projects.

I. Muscalagiu is a university lecturer with a PhD in Computer Science from the Polytechnic University of Timisoara. His research interests include constraint programming and multi-agent system, distributed programming, educational software.

R. Rob, born in 1977, graduated from the Engineering Faculty of Hunedoara, Polytechnic University of Timisoara in 2000. She is currently assistant at the Electrical Engineering and Industrial Informatics Department of Engineering Faculty of Hunedoara, Polytechnic University of Timisoara, Romania.