



DEVELOPMENT OF A CLASS IN THE CONTEXT OF OOP FOR GENERIC MANAGEMENT OF MOUSE EVENTS IN A CANVAS IN THE MATLAB ENVIRONMENT

Emersson Duvan Torres

Luiz Fernando Martha

emerssonto@gmail.com

lfm@tecgraf.puc-rio.br

Pontifícia Universidade Católica do Rio de Janeiro PUC-Rio

Rua Marquês de São Vicente, 225, Gávea, CEP 22451-900, Rio de Janeiro, RJ, Brazil

Abstract. *Teaching of applied computer graphics is of great importance in computational simulation of engineering problems. Currently, many user-friendly computer softwares have improved this work, as it is the case with MATLAB. The generation and manipulation of a geometric model are very important steps in the computational simulation. The use of the mouse allows these steps to become more interactive and easy to understand. For this reason, in this work a generic class is developed in the context of object-oriented programming in the MATLAB environment, which allows managing mouse events in a canvas. The objective of this OOP class is to be used as a base class in the development of graphics and interactive apps in MATLAB, mainly for educational purposes. To meet these expectations, the OOP was adopted, which enables the creation of reusable codes. Allied to this technique, the Unified Modeling Language is used, a graphic language that allows the visualization, construction and documentation of the development of an object-oriented computational system. To determine the correct functioning and practicality of the developed class, two interactive apps are implemented; the first to draw frame structures in 2D and the second to demonstrate the functioning of the Mohr circle for stress state.*

Keywords: *Object-oriented programming (OOP), Unified Modeling Language (UML), Mouse events class, MATLAB*

1 INTRODUCTION

Currently, the increasing evolution of computing has facilitated the resolution of problems in several areas of engineering, becoming an indispensable tool of modeling and analysis. Prior to the existence of advanced computing, these problems had a very time-consuming or even impossible solution because of the high demand for calculations involved in their analysis.

In addition to calculations related to engineering problems, another difficulty that computing often faces is the modeling of these problems and their solutions. Several of these models are too complex to be developed without the aid of computer graphics. At this point the computational modeling process comes to facilitate the solution of existing problems. In this way, it is possible to emphasize the importance of the generation and manipulation of a geometric model. These processes become more reliable and easy to understand through visualization and interaction with the mouse.

The processes of entering data and model manipulation in most programming environments have become less interactive, since commands are usually used to perform these processes; This work intends to change this situation by creating the possibility of using the mouse to perform these activities in apps developed in the MATLAB environment.

To use the mouse in apps developed in MATLAB, in this work a generic class, called Emouse, is developed in the context of Object Oriented Programming (OOP), which allows the management of mouse events. To determine the correct functioning and practicality of this class, two interactive apps are implemented in MATLAB software; The first, e-dles2D (Draw Linear Elements Structure 2D), to draw frame structures in 2D; And the second, e-Mohr2 (Mohr's circle for plane stress state), to demonstrate the operation of the Mohr circle.

Allied to the OOP technique, the Unified Modeling Language (UML) is used. This is a graphical language that allows the visualization, construction and documentation of the development of an object-oriented computational system.

The MATLAB environment is used because it is a tool that, among other factors, has high flexibility and consistency, that allows to develop complex technical calculation apps quickly. In addition, its simple language of easy understanding simplifies the implementation of computational strategies in engineering.

This work is divided into five sections. This first section, in addition to the introduction, presents a small bibliographic review of educational apps developed in the context of OOP in the MATLAB environment. In the second section, the main theoretical concepts used in the development of this work are exposed; such as OO, UML and MATLAB. The third section presents how the Emouse class was developed, with the description of the MATLAB functions used and how it should be implemented in the creation of an app in MATLAB. In the fourth section, the development of two apps that determine the correct operation and practicality of the Emouse class is showed; the first to draw 2D frame structures and the second to demonstrate the functioning of the Mohr circle. The fifth section presents conclusions and suggestions for future work.

1.1 Bibliographic review

This section presents a small bibliographic review of works that develop apps, in the MATLAB environment in the context of OOP, for engineering.

Taking advantage of the context of OOP, web-based and other advanced computing technologies, in Peng (2002) an internet-enabled software framework was developed that facilitates the use and collaborative development of a finite element structural analysis program. The software is designed to give users and developers easy access to the analysis program and the analysis results. In addition, the framework serves as a common finite element analysis platform for which researchers and software developers can build, test, and incorporate new developments. MATLAB is used as a mechanism to build a simple post-processing service, which takes a data file as input and then generates a graphical representation. The MATLAB GUI toolbox and a standard web browser were used to create user interfaces that allows access and analysis of results and project-related information.

The collaborative software developed in Peng (2002), in addition to MATLAB, used Java to represent numerical data, in a convenient way to involve and transmit matrix-type data. In order to incorporate MATLAB into the software structure it is necessary to deal with the communication between the collaborative environments and to develop an object oriented system. MATLAB contains a toolbox that allows to interpret the Java language through its own commands, develop and execute programs that create and access to Java classes and objects. This MATLAB capability enabled the system developed in Peng (2002) to bring Java classes into the MATLAB environment, to construct objects from these classes, to call methods on Java objects, and to save Java objects for later reloading, all performed with MATLAB functions and commands. The OOP provides a layered software architecture that allows to establish a link between MATLAB and Java.

In (Liu et al., 2003) a software framework was developed in the MATLAB environment in the context of OOP, for structural analysis and design research, where different methods of structural analysis and design procedures are studied. This app is designed to be generic for different apps of structural analysis and optimization, design procedures, performance indexes and analysis of methods. It was also implemented with the objective of meeting different types of structural elements of non-linear properties. This software framework is implemented in MATLAB because its language, of easy understanding, allows the use of the OOP and it has the integrated capacity of processing and visualization. The organization that this software obtains from the OOP allows the implementation of new classes in the creation of new apps without the need of changing the general structure. The work presented in (Liu et al., 2003) describes the interactions between the software modules in a UML sequence diagram and the software classes are represented in the UML class diagrams.

An object-oriented approach to structural analysis and its implementation within a finite element software developed in the MATLAB environment are presented in (Harahap et al., 2007). The objective of the work was simplifying the data preparation and analysis cycle, and to find a robust and economical structural configuration.

In (Jankovski et al., 2010) the toolbox *JWM SAOSYS v0.42* (System of Analysis and Structural Optimization) was presented as an experimental prototype toolbox for the MATLAB environment, developed in the context of the OOP. The toolbox is intended for numerical research in the analysis and design of steel structures using the finite element method. In the presentation of this toolbox the main classes and their methods are described in a UML class diagram, each class inherits from a main class that contains the main methods necessary for the operation of the system. In addition to its ease of use, according to (Jankovski et al., 2010), MATLAB features numerous functional and technological facilities that make it an effective tool for designing an

experimental system.

OOP in the *JWM SAOSYS* toolbox allowed its tools to be used for the development of the work presented in (Jankovski et al., 2011), in it the toolbox was updated and a new analysis module (*EPSOp-tim-SD*) was created.

The Finite Cell Method is described in (Zander et al., 2014) as an extension of the finite element method that combines the benefits of higher order finite elements with the fictitious domain idea. In (Zander et al., 2014) the toolbox FCMLab, was developed in the context of the OOP in the MATLAB environment, it was presented as an app and research tool of FCM, which allows the rapid development of new algorithmic methods in the context of domain methods higher order. In this work UML diagrams were used to describe the system components.

In (Kacprzyk et al., 2014) the Isogeometric Analysis was presented as a new formulation in the Finite Element Method, the main objective of this work was the numerical implementation of this method in the MATLAB environment. The OOP context was used to produce a more generic tool for future investigations.

Practical examples of apps developed in the context of the OOP in the MATLAB environment with implementation of graphical interface are presented in (Guardia, 2015) and (Guerrero et al., 2012). There are apps developed in the context of the OOP in the MATLAB in other areas of engineering, for example the software presented in (Elmendorp et al., 2014).

2 CONCEPTS OF OO, UML AND MATLAB

This section presents basic concepts about Object Orientation (OO), Unified Modeling Language (UML) and MATLAB, which were used in the development of this work.

2.1 Object-oriented

Software development with the OO approach consists of building independent modules or objects that can be easily replaced, modified, and reused. It represents the real world view as a collaborative system of objects. In this case, a software is a collection of discrete objects that encapsulate data and operations to model real-world objects. The class describes a group of objects that have similar structures and operations.

The OO approach enables better organization, versatility and reuse of source code, which facilitates software updates and improvements. The OO approach is characterized by the use of classes and objects, and other concepts that will be clarified below.

Classes and Objects

An object represents a real entity, a person, an animal, a house, anything that can be imagined. An object also has characteristics and behaviors, for example, a car has model, type, year of manufacture; As well as behaviors, going, stopping, turning. An object can be identified from the methods and attributes that it has. According to Booch (1991), the objects involved in a system are obtained through the decomposition of the system during the analysis process.

An object can have a state defined as an attribute, for example, a car can have the state "off" or "on", which may be the response to one of its behaviors or methods, for example "turn on".

Classes are species of objects assemblers, which define their characteristics as, the properties and functions that the object possesses. This way of programming allows the user to solve problems using real world concepts.

Booch (1991) defines class as a set of objects that have a common structure and behavior. Also defines that a single object can be called as an instance of a class.

According to (Rumbaugh et al., 2005), an object is a discrete entity with a defined boundary and an identity that encapsulates state and behavior. Also defines class as a collection of objects that share the same attributes, operations, methods, relationships, and behavior.

Thus, according to (Seabra et al., 2015), an object is an abstract entity, which has a boundary defined by its class and meaning for the application. The instance of a class is an object and a class represents an abstraction of the similar characteristics of a given set of objects.

Attributes and Methods

A class contains attributes and methods, which are, respectively, characteristics and routines that can be executed by an object of this class. An instance can be differentiated from the others by the combination of attributes and methods, which define its identity, its state and behavior. Methods and attributes of a class are defined during system design. For example, with the "car" class you can create a "jeep" object, which has the "color" attribute, the "carry" method.

According to (Rumbaugh et al., 2005), an attribute is the description of a repository of a given type of data in a class. Each object can have a certain value for the attribute. A method is the implementation of an operation. It specifies the algorithm or procedure that results from the operation.

Attributes are local variables that store values of an object's characteristics. Methods are the activities, actions, or operations performed by a class. A method can receive parameters and return values. Returns values can indicate the success of the operation or the resulting value.

Thus, methods are the functions that an object can perform and attributes are everything an object has as a variable.

Inheritance and Polymorphism

Inheritance is a feature that allows to given class to inherit the characteristics of another class, called parent class or superclass. The descendant class, called the child class, acquired all methods and attributes of the parent class.

According to O'Docherty (2005), the importance of inheritance in implementation is summarized in:

- Inheritance supports a richer and more powerful modeling. It benefits the entire development team as it provides greater code reuse power.
- Inheritance enables define information and behaviors in one class and share them with others. This results in less code to write.
- Inheritance is natural. This is one of the most important reasons for OO to be in the first place.

Inheritance can be single or multiple. It is single when a class inherits from only one superclass, however, it is multiple when inherits from several classes. This last one, although, is little diffused and of complex use, its implementation in MATLAB is possible, serving as another benefit to the programmer.

The polymorphism is a feature of OO which means the ability to modify methods inherited from a superclass. That is, by inheriting the attributes or methods from a superclass, these can be redeclared in the new class. In a purely OO language, all non-private variables and all methods can be modified through this feature.

Encapsulation is another concept used in OOP, this is the act of hiding to the user the internal processes of an object, class or method.

2.2 Unified Modeling Language

The unified modeling language (UML) according to (Booch et al., 1998) is a standard graphical language for visualizing, specifying, constructing, and documenting OO systems. This very expressive OO modeling language addresses all the views needed to develop and deploy the software system. With its embedded vocabulary, graphic symbols, and rules, the UML simplifies the complex process of software design and facilitates communication with an explicit model.

According to (Weilkiens et al., 2006), the UML can be defined with a metamodel, the prefix "meta" is used because the language is defined as level below the user's level of utilization, that is, the specification of this language is performed with the proper use of the language.

According to Booch (1991), the vocabulary of UML includes three components: items, relationships, and diagrams. The UML items are classified into four categories:

- Structural items, are the nouns of UML models, such as a class, rendered as a rectangle graphically, usually including its name, attributes, and operations.
- Behavioral items, are the dynamic parts of UML models to represent the behavior over time and space. An interaction is a behavior that comprises a set of messages exchanged among a set of objects within a particular context to accomplish a purpose. Graphically, a message is rendered as a directed line, including the name of the operation.
- Grouping items, are the organizational parts of UML models. One example is the package.
- Annotational items, are the explanatory parts of UML models. A primary example is a note that comments about the element in a model. It is rendered as a rectangle with a dog-eared corner, together with a comment.

Classes relate to each other, allowing information sharing and the association of methods to accomplish specific tasks. According to the purpose of the relationship, there are four kinds of relationships in the UML: dependency, association, generalization, and realization (Weilkiens et al., 2006), as described below:

- Association: describes a set of links, that is, connections among objects. Graphically, an association is rendered as a solid line, possibly directed. This relationship have a subclassification:

- Aggregation, representing an independence relationship between a whole and its parts. In the destruction of the whole, the parts can still exist. Graphically, is rendered as a solid line with a hollow diamond on the container class side.
- Composition, representing a dependence relationship between a whole and its parts. In the destruction of the whole, the parts no longer make sense and cease to exist. Graphically, is rendered as a solid line with a solid diamond on the container class side.
- Generalization: a (inheritance) relationship in which objects of the specialized class, the child, are substitutable for objects of the generalized class, the parent. A generalization relationship is rendered as a solid line with a hollow arrowhead pointing to the parent.
- Dependency: a semantic relationship between two elements, where a change to one element may affect the semantics of the another element, the dependent element. Graphically, a dependency is rendered as a dashed arrowhead pointing to the independent element.
- Realization: a semantic relationship between classes where one class specifies a contract and another class guarantees to carry it out. A realization relationship is rendered as a dashed line and a hollow arrowhead.

The graphical representation of the UML items and relationships is shown in Figure 1.

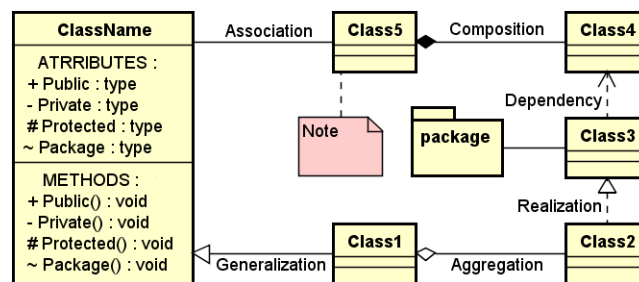


Figure 1: UML items and relationships (Liu, 2003).

A UML diagram is the graphical presentation of a set of elements, often rendered as a connected graph of vertices, items and arcs, relationships. The UML includes several kinds of diagrams. The most common kind is a class diagram that shows a set of classes, interfaces, their collaborations and relationships. An activity diagram, shows the stages or steps for completion of a particular activity in the system. An interaction diagram consists of a set of objects and their relationships, including the messages that may be dispatched among them. There can be two types of interaction diagrams: a sequence diagram emphasizes the time-ordering of messages and a collaboration diagram emphasizes the structural organization of the objects that send and receive messages.

2.3 MATLAB

MATLAB is a numerical analysis and data visualization software with graphical and programming capabilities. Although its name means Matrix Laboratory, its purposes are currently wider. This software was created as a program for mathematical operations with matrices, but later it became a very useful and flexible computational system. It includes an integrated development environment, such as object-oriented programming constructs.

Its language is based on a simple mathematical language, making its work environment easy to use. Thus MATLAB is a tool and a programming language of high level, and has as main functions: construction of graphs and compilation of functions, manipulation of specific functions of calculation and symbolic variables. In addition, this software has built-in functions to perform many operations, and a large number of auxiliary libraries (Toolboxes) that can be added to enhance these functions.

OOP in MATLAB

OOP is a formal programming approach that combines data and associated actions (methods) into logical structures (objects). This approach increases the ability to manage the complexity of software when developing apps and structures that use data with very large sizes.

MATLAB's OOP capabilities allows develop complex technical calculation apps at high speed. It is possible to define classes and apply OO design patterns in MATLAB that allow reuse of code, inheritance, encapsulation, and reference behavior without having to pay attention to the usual low-level tasks required in other languages.

MATLAB OOP involves the use of:

- Class definition files, which allow the definition of properties, methods, and events.
- Classes with reference behavior, which help to create data structures such as linked lists.
- Transmitters and receivers, which allow to monitor the actions and changes of the object properties.

GUI MATLAB

GUIs, also known as graphical user interfaces or user interfaces, allows simple control of software app developed in MATLAB to eliminate the need of writing commands in order to interact with an app.

MATLAB apps are self-contained MATLAB programs with a graphical GUI user interface that automates a task or a calculation. A GUI typically contains controls such as menus, toolbars, buttons, and sliders. MATLAB allows to create owns apps with custom user interfaces to allow other users to use them.

GUIDE (GUI development environment) provides tools for designing user interfaces for custom app. It is possible to graph the users interface using the GUIDE design editor. Then, GUIDE automatically generates a MATLAB code to construct the interface, this code can be modified to program the behavior of the app.

It is possible to create MATLAB code that defines all component properties and behaviors in order to have more control over the design and development. MATLAB contains a built-in functionality that allows to create the GUI for an app programmatically. Also, it has the ability to add dialog boxes, user interface controls such as buttons and sliders, and containers such as panels and button groups (GUI components).

Callback Functions

Each type of GUI component has actions that can be exercise on it. It is possible to associate each of these actions with a callback function. A callback function is defined as the response to an action that a GUI component will perform when the user activates it.

When a GUI is created in MATLAB, the GUIDE environment automatically generates a file with the callback functions related to the principals events that can occur with each GUI component of the interface.

3 EMOUSE CLASS

The Emouse is an abstract class, developed in the MATLAB environment, to facilitate the development of applications that handle mouse events on canvas (axes: the drawing area of a GUI application in MATLAB).

The abstract Emouse class presents, in addition to the constructor method, four private concrete methods (implemented) and three abstract methods that must be implemented by the client user. Its use is achieved by creating a client subclass that inherits its properties and implements the three abstract methods.

3.1 *figure* and *axes* Objects

When a figure is made in MATLAB, two objects are created. The first one called *figure*, is the window where the results are drawn, this object has properties like color, name, position, etc. Of these attributes the most important for the development of the Emouse class are:

- **CurrentPoint**, current position of the mouse in the axes of the *figure* object. This attribute is used in the *eMouseMove* concrete method of the Emouse class.
- **SelectionType**, attribute that provides information about the last mouse button pressed inside the figure window. This information indicates the type of selection made: right, left or center button. This attribute is used in the *eButtonDown* concrete method of the Emouse class.

The object *figure* also presents some methods, among these methods the most important, for the development of the class Emouse, are:

- **WindowButtonMotionFcn**, callback function that is executed whenever the user moves the mouse inside the figure window.
- **WindowButtonDownFcn**, callback function that is executed whenever the user presses a mouse button inside the figure window.
- **WindowButtonUpFcn**, callback function that is executed whenever the user releases a mouse button.

These callback functions are implemented in the constructor method of the Emouse class.

The second object created when a figure is made in MATLAB is called *axes*, which contains the properties of the drawing space (canvas) that is inside the figure window. This object has attributes such as the axis limit, font size, background color etc. Among these attributes the most important, for the development of the class Emouse, is:

- **CurrentPoint**, current position of the mouse in the axes of the *axes* object. This attribute is used in the *eMouseMove* concrete method of the Emouse class.

3.2 Attributes of the Emouse class

The Emouse class has the following attributes, which store the information that is obtained with a mouse event:

- `dialog`: contains a *figure* object associated to mouse events.
- `canvas`: contains a current *axes* object associated to mouse events.
- `mouseButtonMode`: defined as a variable of type string, but that works like a boolean, because it presents two states: 'up' when the mouse buttons are not pressed, and 'down' when one of the mouse buttons is pressed. This attribute is initialized in the 'up' state.
- `whichMouseButton`: determines which of the mouse buttons was pressed, it is defined as a string variable, displays the following states: 'left' to the left button, 'right' to the right button, 'center' to the center button, and 'none' when none button is pressed. This attribute also defines whether the user presses the left button twice consecutively with the 'double click' state. This attribute is initialized in 'none' state.
- `currentPosition`: contains the current position of the mouse on the canvas; it is defined as an array of 1x2 order, where the variable 1x1 presents the X coordinate and the variable 1x2, the Y coordinate.

3.3 Methods of the Emouse class

The Emouse class has the following methods, which update its attributes to manage mouse events:

- Constructor method, intended to initialize an object of this class, its input parameters are a *figure* object and an *axes* object. This method sets the units property of the target figure (`dialog`) to pixels and associates the mouse button down, mouse move, and mouse button up events on the target figure with the private `eButtonDown`, `eMouseMove`, and `eButtonUp` methods, respectively.

For the development of the Emouse class, three events were considered:

- When the mouse is in motion, the `eMouseMove` method is called. This event is implemented by using the following line of code:

```
set(this.dialog, 'WindowButtonMotionFcn', @this.eMouseMove);
```

- When a mouse button is pressed, the `eButtonDown` method is called. This event is implemented by using the following line of code:

```
set(this.dialog, 'WindowButtonDownFcn', @this.eButtonDown);
```

- When the mouse button is released, the `eButtonUp` method is called. This event is implemented by using the following line of code:

```
set(this.dialog, 'WindowButtonUpFcn', @this.eButtonUp);
```

- `eButtonDown`: this method is a callback function associated with mouse button down event on the target canvas. It finds, in the list of axes (canvases) of the target figure (`dialog`), the axes (canvas) in which the button down position is located. The method also determines which button was pressed, updates the `whichMouseButton` property with this

information, sets the `mouseButtonMode` property to 'down', sets the current position to the mouse button down position, and calls the abstract `downAction` method.

- `eMouseMove`: this method is a callback function associated with mouse move event on the target figure (dialog). It sets the current position to the current mouse position on the target axes (canvas) and calls the abstract `moveAction` method.
- `eButtonUp`: this method is a callback function associated with mouse button up event on the target figure (dialog). It sets the `mouseButtonMode` property to 'up', sets the current position to the mouse button up position on the target axes (canvas), and calls the abstract `upAction` method.
- `clean`: this method clears the attributes of the class and assigns them the initial data.

The `Emouse` class also has the following three abstract methods that must be implemented by the client user:

- `downAction`: this method must be implemented with the procedures to be performed when the user presses a mouse button.
- `moveAction`: this method must be implemented with the procedures to be performed when the user moves the mouse.
- `upAction`: this method must be implemented with the procedures to be performed when the user releases the mouse button that was pressed.

3.4 Use of the `Emouse` class

To use the `Emouse` class, follow the steps below:

1. Create a subclass that inherits from the `Emouse` class.
2. Implement new attributes in the properties of this subclass if they are required for the app development, for example, an attribute that counts the number of times a mouse button has been pressed.
3. Implement a constructor method for this subclass that initializes the inheritance of the `Emouse` class. This method must have as input arguments a *figure* object and an initial *axes* (canvas) object that should be supplied to the `Emouse` class in the inheritance.
4. Implement the abstract methods of the `Emouse` class in this subclass. New methods can be implemented in this subclass, if they are necessary for the app, for example, a method that incorporates the use of a keyboard button.
5. When it is necessary to use the mouse events, create an object of the new subclass and providing a *figure* object and an initial *axes* object that will be associated with the mouse events.

4 APPS DEVELOPED WITH THE EMOUSE CLASS

As examples of use and to determine the correct functioning of the `Emouse` class, two apps, that need the management of mouse events, have been developed; The first, named `e-dles2D`, is used to draw frame structures in 2D and the second, called `e-mohr2`, to demonstrate the behavior of the Mohr circle for plane stress state.

In these two apps the MATLAB inpolygon function is used, which determines whether a point is located inside or on the edge of a polygonal region. Its input parameters are the coordinates of the point, and two vectors (x and y) with the coordinates of the points that make up the polygonal region. A boolean is obtained as the output parameter, 'true' if the point is inside the polygon and 'false' otherwise.

4.1 e-dles2D

The e-dles2D (Draw Linear Elements Structure 2D) app allows to draw frame structures in 2D, it has two types of main entities, nodes and linear elements, the latter representing bar elements.

In the development of this app were created nine classes and one subclass, the following is a short description of each of the classes implemented. A simple GUI has also been implemented, that allows to execute objects from the classes that initialize the app, and control the functionality of GUI entities.

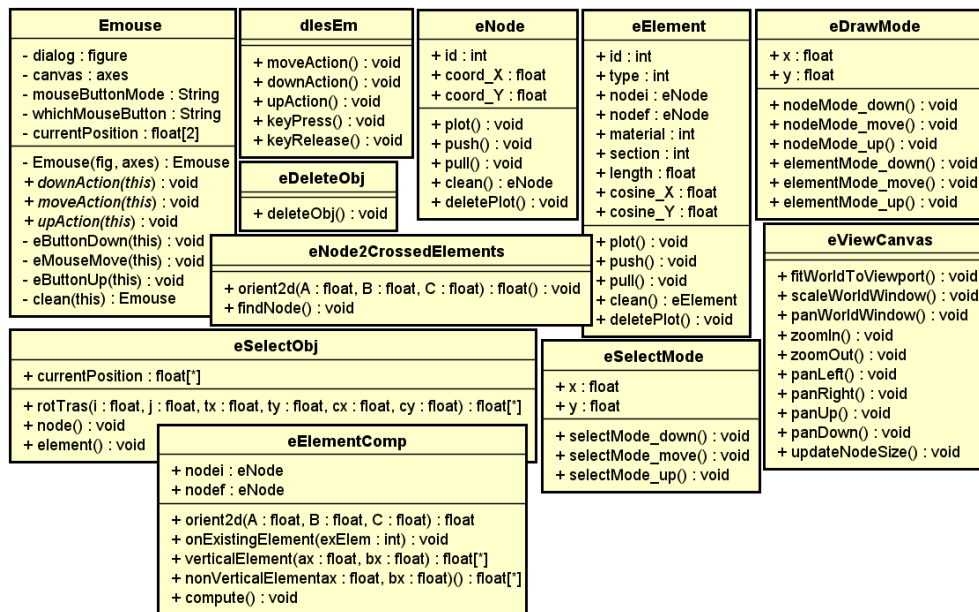


Figure 2: Definition of attributes and methods of the e-dles2 app classes.

- dlesEm: this subclass inherits the attributes of the Emouse class and implements its abstract methods of mouse events for the purpose of the app.
- eNode: this class allows to instantiate objects that represent nodes of a frame structure, its main methods are to draw and to delete a node of the canvas.
- eElement: this class allows to instantiate objects that represent bars of a frame structure, its main methods are to draw and to delete a bar of the canvas.
- eDrawMode: this class contains the app draw modes, presents methods that draw, with the mouse, the entities (nodes or bars) according to the actions performed with the mouse.
- eElementComp: this class allows defining an object to create and draw bars on the canvas, taking into account any modifications that may occur with the bars due to very close or collinear entities.

- **eSelectMode**: this class contains the entities selection modes on the canvas, presents methods that allow to select existing entities on the canvas with the mouse.
- **eDeleteObj**: this class contains no attributes, and contains a single method, in addition to the constructor, that allows to delete selected entities on canvas.
- **eSelectObj**: this class allows to recognize if the mouse is near to an entity on the canvas, and find the midpoint of a bar type entity.
- **eNode2CrossedElements**: this class allows to create a node at the intersection of two bar elements.
- **eViewCanvas**: this class allows to manipulate the canvas, moving it in the horizontal or vertical direction, increasing or decreasing the zoom and adjust it so that the entire drawn frame structure is visible on the canvas.

The definition of attributes and methods and the relationship of the previous classes are presented in Figures 2 and 3, respectively.

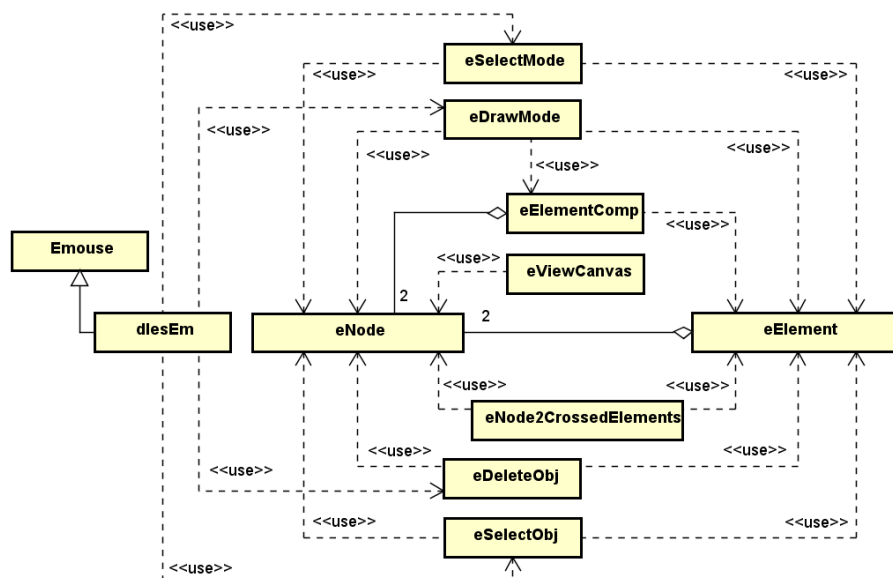


Figure 3: Class diagram of the e-dles2D app.

Graphical interface of the e-dles2D app

Figure 4 shows the graphical interface developed for the e-dles2D app. The following is described the operation of this interface:

Node button (1), allows to create and drawing a node on canvas at coordinates clicked with the mouse.

Element button (2), allows to create and drawing a bar element on canvas, selecting two points on canvas with the mouse.

Select button (3), allows to select, with the mouse, one or more entities drawn on canvas. It is possible accumulate selected entities by holding the shift button on the keyboard, or also by creating a rectangle with the mouse that surrounds the desired entities.

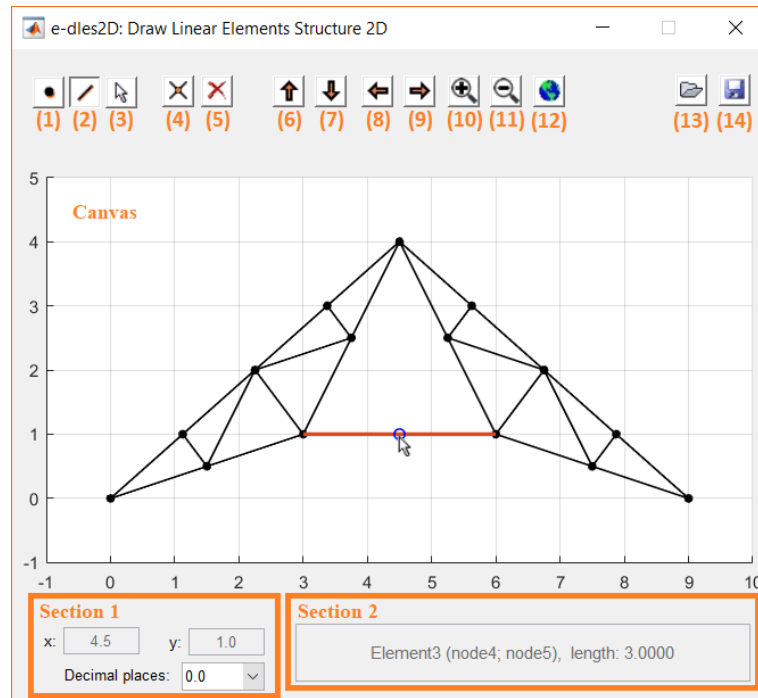


Figure 4: Graphical interface of the e-dles2D app.

Intersection bar button (4), allows to find a node at the intersection of two selected intersected bar elements.

Delete button (5), allows to delete the selected entities from the memory and erase them from the canvas.

Pan up button (6), allows to move the image on canvas upwards.

Pan down button (7), allows to move the image on canvas downwards.

Pan left button (8), allows to move the image on canvas to the left.

Pan right button (9), allows to move the image on canvas to the right.

Zoom in button (10), allows to decrease the portion of the canvas view.

Zoom out button (11), allows to increase the portion of the canvas view.

Fit world button (12), allows to adjust the canvas so that the drawing is visible in its entirety.

Open button (13), allows to open an e-dles2D model.

Save button (14), allows to save an e-dles2D model.

Section 1, presents the x and y textity coordinates of the current position of the mouse on canvas. In addition, it contains the decimal places menu, which lets select the number of decimal places of these coordinates. For example, Figure 4 shows that the mouse is near to the center point of element 3, with the coordinates (4.5, 1.0) with one decimal place.

Section 2, shows the length of a bar element being drawn. Also displays the name and geometric properties of the entity that is below the mouse. For example, Figure 4 shows the properties of element 3, which is below the mouse.

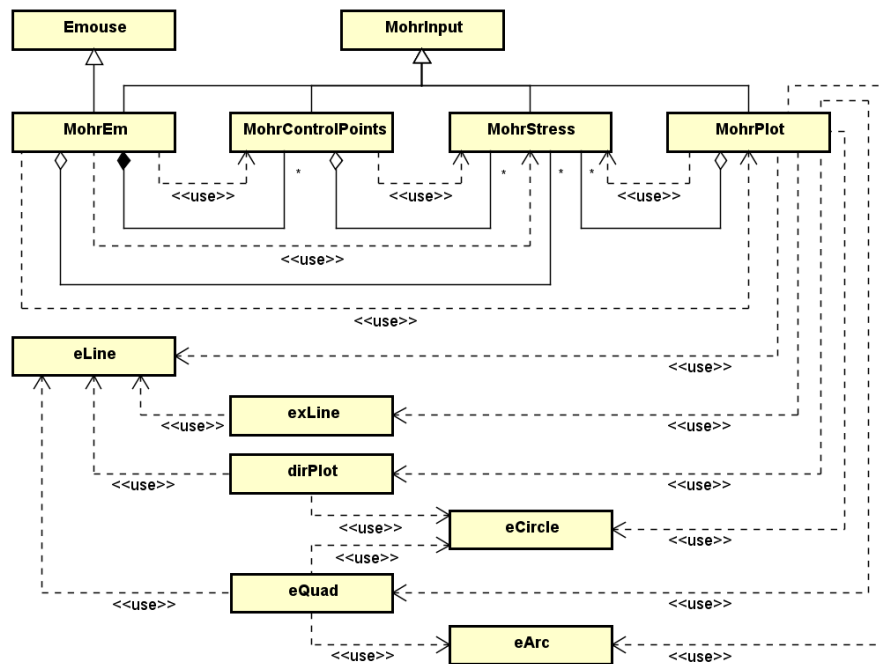


Figure 6: Class diagram of the e-Mohr2 app.

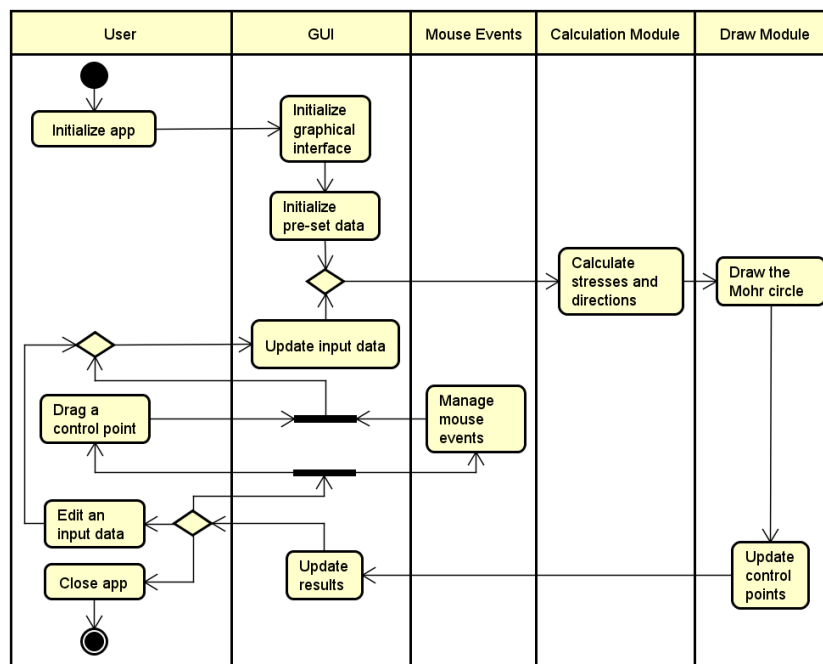


Figure 7: Activity diagram of the e-Mohr2 app.

- MohrControlPoints: this subclass inherits the attributes of the MohrInput class, and it has methods that allow to determine if the mouse is near to a the Mohr circle control point.
- MohrPlot: this subclass inherits the attributes of the MohrInput class, and it has methods that allow to draw the Mohr circle on canvas.
- dirPlot: this class has the methods that allow to draw the arrows that define the direction of a certain stress acting on the Mohr circle.

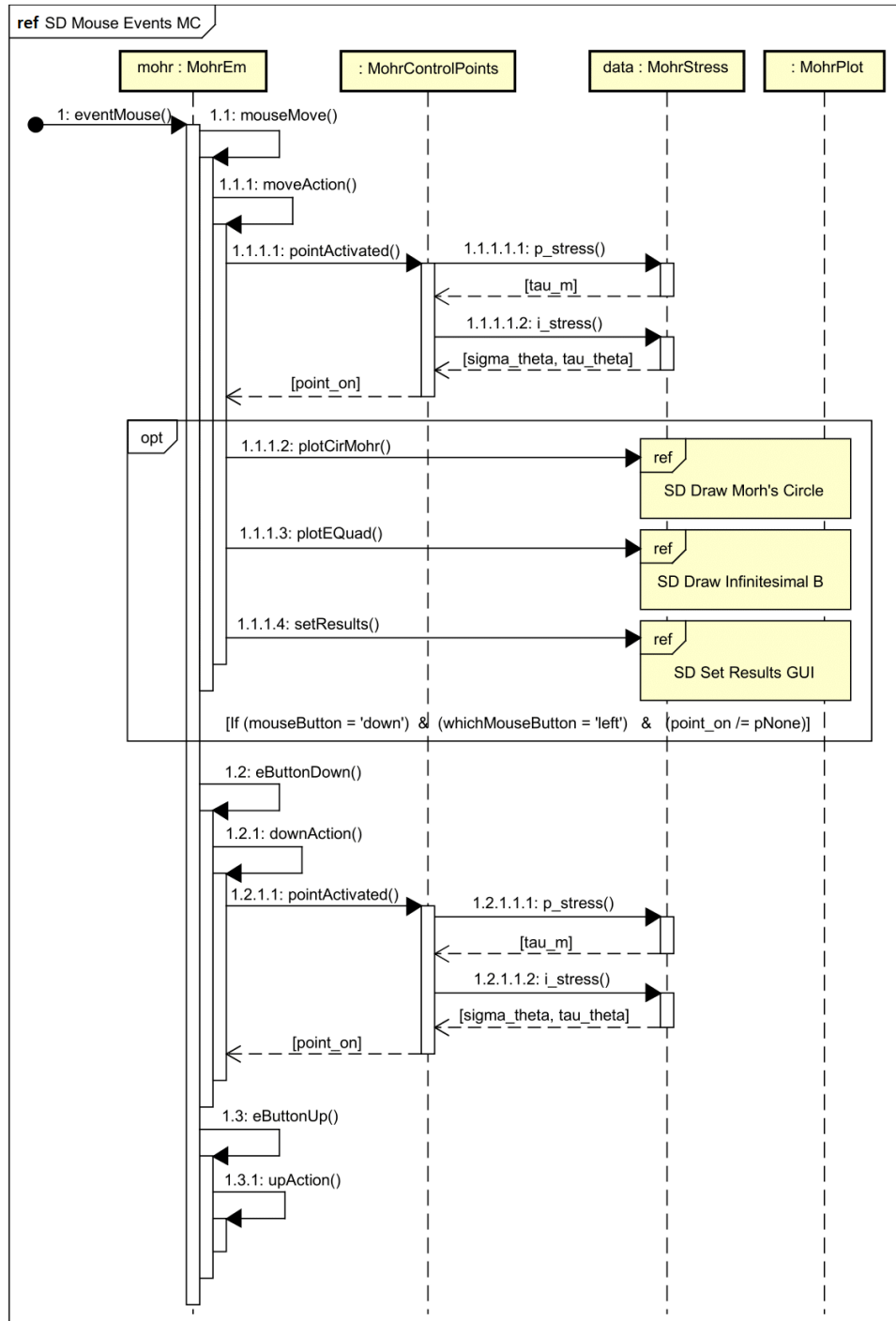


Figure 8: Fragment (SD Mouse Events MC) of the sequence diagram of the e-Mohr2.

- eQuad: this class has the methods that allow to draw the stresses of the Mohr circle acting on an infinitesimal element according to the inclination plane.

The definition of attributes and methods and the relationship of the previous classes are presented in Figures 5 and 6, respectively.

Figure 7 shows the activity diagram of the e-Mohr2 app. This diagram allows to observe the actions and the communication developed by five actors: the user, the graphical interface

(GUI), a mouse event manager module, a calculation module and a draw module. These last three actors represent the classes described above. This diagram provides an overview of the operation of the app and its interaction with the user.

The sequence diagrams obtained for the developed apps are very extensive, for this reason and to illustrate the implementation of the abstract methods of mouse events of the class Emouse by a subclass, only a fragment of the sequence diagram of the e-Mohr2 app is presented in Figure 8, where the MohrEm subclass performs a message exchange during the operation of the app.

Graphical interface of the e-Mohr2 app

Figure 9 shows the graphical interface developed for the e-Mohr2 app. The following is described the operation of this interface:

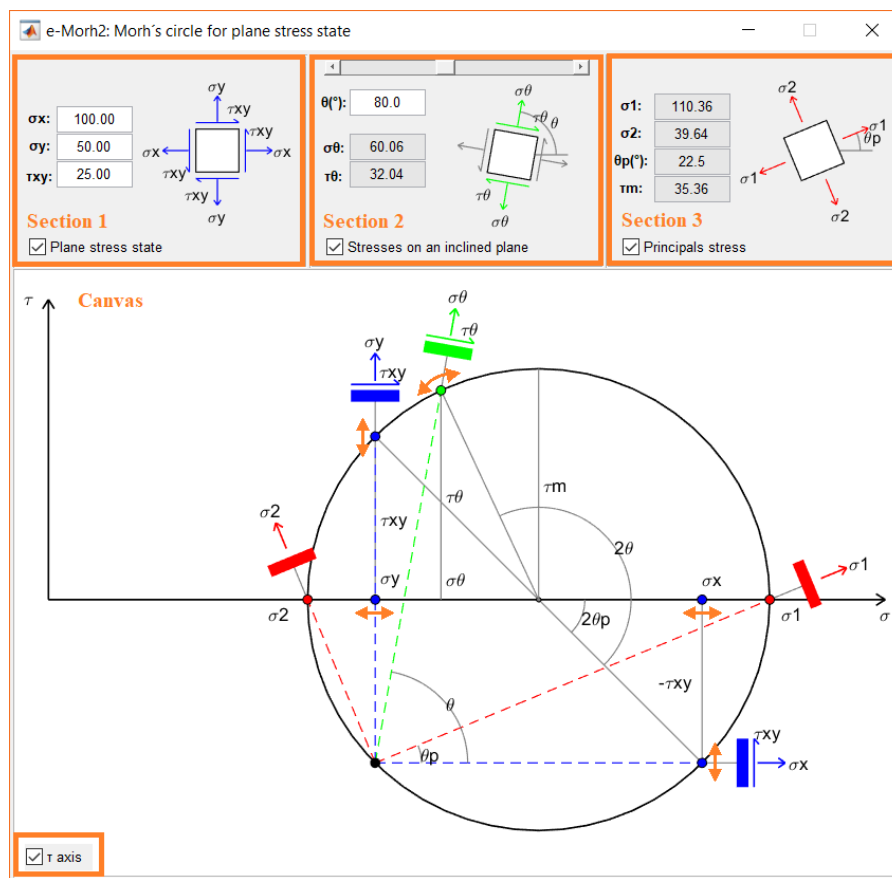


Figure 9: Graphical interface of the e-Mohr2 app.

Section 1, allows to edit the input data σ_x , σ_y and τ_{xy} . In addition, it contains the plane stress state checkbox, that allows to show or not show the plane state stresses on canvas. In this section also is plotted the plane state stresses of the Mohr circle acting on an infinitesimal element.

Section 2, allows to edit the angle of inclination of the action plane of the stress θ , in degrees, by entering text or by means of a slide bar. In addition, it displays the output data σ_θ and τ_θ . And it contains the checkbox stresses on an inclined plane, that allows to show or

not show the stresses that act on the inclined plane on canvas. In this section an infinitesimal element is also plotted with the stresses acting on an inclined plane of the Mohr circle.

Section 3, presents the output data σ_1 , σ_2 , θ_p , in degrees e τ_m . In addition, it contains the principal stress checkbox, that allows to show or not show the principals stresses on the canvas. In this section also is plotted the principal stress of the Mohr circle acting on an infinitesimal element.

In the lower-left corner, the interface presents the checkbox τ axis, that allows to show or not show the τ axis on the canvas.

Figure 9 allows to observe the control points of the Mohr circle. The points $(\sigma_y, 0)$ and $(\sigma_x, 0)$ (blue color) can be dragged with the mouse to the right or to the left. The points (σ_y, τ_{xy}) and $(\sigma_x, -\tau_{xy})$ (blue color) can be dragged with the mouse up or down. And finally, the point $(\sigma_\theta, \tau_\theta)$ (green color) can be dragged with the mouse around the circle of Mohr. When one of these points is modified, the changes that occur are displayed on the canvas and the GUI.

5 CONCLUSIONS

A class in the context of OOP for generic management of mouse events on a canvas in the MATLAB environment was developed. Using this class, two engineering apps, that need the management of mouse events on a canvas, were developed in the MATLAB environment in the context of OOP. The development of these apps highlights the generic property of this class.

The OOP context was employed in this work, because it provides a simple and reusable maintenance code that favors the creation of new functionalities. When an OO system is developed, the UML can improve its understanding by providing graphical documentation of the behavior of the system.

The developed class can be used at the implementation of MATLAB apps, in the context of OOP, besides visualization, creation, and manipulation of models, aiming to be educational graphic-interactive tools. As a result of the mouse interaction these processes become quite better comprehended.

The proposal for future work includes: implementing the events that occur with the mouse scroll button; Implement mouse events that consider three dimensions; Create a more complete version of the e-dles2D app, where can be set boundary conditions and geometric and constitutive properties of the elements.

However, the future work has a wider view, because the class obtained is a potential tool to develop, in the MATLAB environment, educational graphic-interactive softwares.

ACKNOWLEDGEMENTS

The authors are grateful to the Department of Civil Engineering of the PUC-Rio for their support to research. The first author is grateful to CAPES for the financial support.

REFERENCES

Booch, G., 1991. *Object-oriented Analysis and Design with Applications*. Addison-Wesley.

- Booch, G., Rumbaugh, J., & Jacobson, I., 1998. *The unified modeling language user guide*. Addison-Wesley.
- Elmendorp, R., Vos, R., & La Rocca, G. 2014. A conceptual design and analysis method for conventional and unconventional airplanes. In: ICAS 2014, *Proceedings of the 29th Congress of the International Council of the Aeronautical Sciences, St. Petersburg*.
- Guardia, J., 2015. Desarrollo en Matlab de software de cálculo de placas por MEF. *Universitat Politècnica de Catalunya*, Barcelona.
- Guerrero, L., Pizano, D., & Thomson, P., 2012. Desarrollo e implementación de una interfaz gráfica para el programa de elementos finitos FEM. In: *22th Jornadasaie*.
- Harahap, I., Jr, V., & Mustapha, M., 2007. A new approach of structural analysis in conjunction with structural design and optimization. *Mission-Oriented Research: PETROCHEMICAL CATALYSIS TECHNOLOGY*, vol. 5, n. 2, pp. 32-42.
- Jankovski, V., & Atkočiūnas, J., 2010. Saosys toolbox as Matlab implementation in the elastic-plastic analysis and optimal design of steel frame structures. *Journal of Civil Engineering and Management*, vol. 16, n. 1, pp. 103-121.
- Jankovski, V., & Atkočiūnas, J., 2011. Biparametric shakedown design of steel frame structures. *Mechanics*, vol. 17, n. 1, pp. 5-12.
- Kacprzyk, Z., & Ostapska-Łuczkowska, K., 2014. Isogeometric Analysis as a New FEM Formulation-Simple Problems of Steady State Thermal Analysis. *Procedia Engineering*, vol. 91, pp. 87-92.
- Liu, W., Tong, M., Wu, X., & Lee, G., 2003. Object-oriented modeling of structural analysis and design with application to damping device configuration. *Journal of computing in civil engineering*, vol. 17, n. 2, pp. 113-122.
- Martha, L., Carbone, A., Pereira, A., Ramires, F., Dalcanal, P., & Rodrigues, R., 2004. *e-Mohr: Ferramenta educacional para círculo de Mohr*. [software, v. 1.0]. Projeto Final CIV2802, Departamento de Engenharia Civil, Pontifícia Universidade Católica do Rio de Janeiro.
- O'Docherty, M., 2005. *Object-oriented analysis and design: Understanding system development with UML 2.0*. John Wiley Sons Ltda.
- Peng, J., 2002. *An Internet-enabled software framework for the collaborative development of a structural analysis program*. PhD thesis, Stanford University/California.
- Rumbaugh, J., Jacobson, I., & Booch, G., 2005. *The Unified Modeling Language Reference Manual*. Pearson Education.
- Seabra, J., 2013. *UML - Unified Modeling Language: Uma ferramenta para o Desing de Software*. Editora Ciência Moderna Ltda.
- Weilkiens, T., & Oestereich, B., 2006. *UML 2 Certification Guide - Fundamental Intermediate Exams*. Morgam Kaufmann Publishers.
- Zander, N., Bog, T., Elhaddad, M., Espinoza, R., Hu, H., Joly, A., ... & Parvizian, J., 2014. FCMLab: A finite cell research toolbox for MATLAB. *Advances in engineering software*, vol. 74, pp. 49-63.