Université de Lille

cea

École doctorale régionale Sciences Pour l'Ingénieur Lille Nord-de-France

Thèse praparée pour obtenir le grade de

Docteur de l'Université de Lille

Discipline : Informatique et applications

Soutenue publiquement par

Yosser EL AHMAR

Le 06 Décembre 2018

Améliorer l'efficacité cognitive des diagrammes UML : application de la Sémiologie Graphique

Enhancing the Cognitive Effectiveness of UML Diagrams: Application of the Semiology of Graphics

Directeur de thèse : **Xavier LE PALLEC**
Co-directeur de thèse : **Sébastien GÉRARD**

Jury

Kari SYSTÄ	Professeur, Université de technologie de Tampere, Finlande	Rapporteur
Michel R.V. CHAUDRON	Professeur, Université de Gothenburg, Suède	Rapporteur
Monique NOIRHOMME	Professeur, Université de Namur, Belgique	Examinatrice
Sophie DUPUY-CHESSA	Professeur, Université de Grenoble Alpes, France	Examinatrice
Jean-Sébastien SOTTET	Chercheur, Institut des sciences et technologies, Luxembourg	Examinateur
Jordi CABOT	Professeur, Université ouverte de Catalogne, Espagne	Examinateur

Université de Lille
Centre de Recherche en Informatique, Signal et Automatique de Lille - CRIStAL.
Commissariat à l'Énergie atomique et aux énergies Alternatives - CEA de Saclay
Laboratoire d'Integration des Systèmes et des Technologies - LIST.

Acknowledgments

First of all, I would like to express my sincere gratitude to Xavier LE PALLEC who supervised this thesis. Xavier gave me the opportunity to do this research work and created an ideal context for it. In addition to his support, advice and knowledge he transmitted to me, I would like to thank him for his availability and encouragement.

I am also extremely grateful to Sébastien GÉRARD for his wise advice, his pertinent remarks and his scientific rigor.

My sincere thanks goes to Kari SYSTÄ and Michel R.V. CHAUDRON who have done me the honor to evaluate my work and to be the referees. I would like to thank them for their constructive comments.

I would like to especially thank Michel R. V. CHAUDRON for welcoming me to his laboratory as a visiting student. He gave me the opportunity to acquire knowledge about empirical research. His insightful and timely remarks have had a positive impact on my research.

I also thank Monique NOIRHOMME, Sophie DUPUY-CHESSA, Jean-Sébastien SOTTET and Jordi CABOT for having accepted to be part of the jury members and for their interesting comments and challenging questions.

The work presented in this thesis has been financed by the CEA and done within the Laboratory for Integration of Systems and Technology (LIST). So thanks again to Sébastien GÉRARD for hosting me in his laboratory.

A big thanks is addressed to all my colleagues, members of the LIST and the CARBON teams for the very favorable atmosphere they created around me. In particular: Camille, Fadoua, Sahar, Slim, Michel and Mickaël.

A special thanks to all my friends, who recognize themselves, who supported me during the difficult times, especially my dear Narjes.

The totality of my gratitude and my thoughts go to my parents Mouhamed and Karima, to my brother Oussema and my sister Ghada. They never stopped encouraging me and giving me support. It is therefore natural that this document is dedicated to them.

And, of course, I express my deepest gratitude for Houssem, my husband, who has been able to bear me and to cheer me up with infinite patience during these years.

Finally, a big hug to my dear son Iyed who brought light and happiness to my life some weeks before the defense of this thesis.

Antony, 01-25-2019

Contents

List of Figures

List of Tables

List of Abbreviations

CDs Cognitive Dimensions

PoNs Physics of Notations

SoG Semiology of Graphics

UML Unified Modeling Language

Chapter 1

Introduction

Contents

1.1 Context

Graphical representations are a key element in software engineering. If textual representations are efficient for linear reasoning, graphical ones score higher to show links among a set of elements [31]. But, the effective use of such visual representations requires to deeply understand how they work.

According to Jaques Bertin [10], graphical representations have three functions. The first one is: saving information. It consists of drawing an inventory to save all information describing a particular situation or elements constituting a system (already existing or future one). Therefore, the reader maintains a sort of memory in which she/he can navigate to find relevant information for her/his task. This type of inventory may for sure serve to communicate information between two or more than two stakeholders. In fact, if the author assumes that some information is more relevant for the task that the receivers have to accomplish, she/he can highlight it (e.g., by increasing the volume of the sound of an audio representation or by increasing the size of a spot in a graphical representation). Communication is the second elementary function of a representation. The third function is the treatment of information. Some tasks like the optimization of a system or the determination of the behaviour of a kind of customers need to infer information. By organizing in different manners the information contained in a representation, readers may perceive (accidentally or not) correlations that might form new knowledge that is valuable for their activity [38]. The reorganization task represents a way to help humans to treat information. The techniques of visualization of information that are mostly employed nowadays in Big Data / Business Intelligence have proven the effectiveness of the graphical representation for many years in that objective.

For these three functions, the graphical system has a major benefit compared to linear systems like the audio system or the textual one: it allows readers to rapidly identify links between the represented elements [31]. But it also has the capacity to instantly present multiple properties of an information with each displayed spot or mark: via its position[1] and via its visual aspect (color, shape, size, orientation, grain and brightness). Each one of these means of visual variation allows to indicate different values for a particular property. X and Y axis, color, shape, size, orientation, grain and brightness are called visual variables. The visual variables except the X and Y planar axis are called retinal variables [10]. They are all depicted in Figure 1.1.

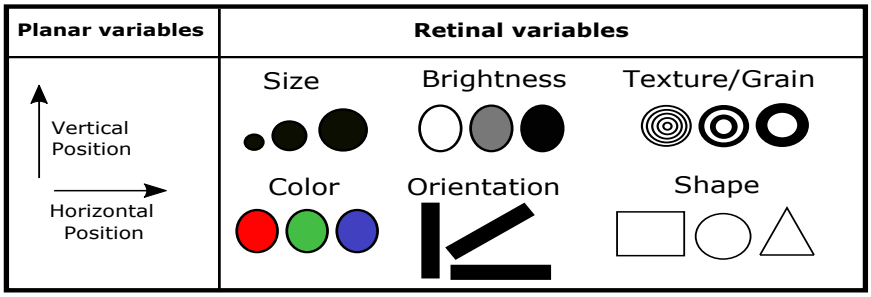

Figure 1.1: The visual variables

The importance of the visual variables comes from their significance in the depth perception [10]. Experimental psychology defines the perception in a third dimension

[1] in correspondence with the information in the plan (X or Y can indicate, each one, a value for a particular property)

as the result of multiple facts. First, the binocular vision (i.e., using two eyes) in a limit of six meters. Second, the apparent movement of an object when the reader moves. Third, the decrease of the size of a known object. Fourth, the decrease of the brightness of a known contrast. Fifth, the reduction of the grain of a known texture. Sixth, changing the known color of an object. Finally, the deformation of the shape and orientation of a known object. All the previously mentioned variations except from the two first ones are available to graphic authors to put information in a third dimension. The retinal variables are relatively rapidly perceived because the reader's eye can detect their variation without moving the visual brush, the signals which are received on the retina are sufficient. In fact, the retinal variables are pre-attentively treated (without a conscious attention) by an automatic, very fast, and mostly parallel dedicated system: the visual system. Then, the retinal variables that captured the attention of the reader are cognitively processed by accessing to the short term memory and/or the long term memory. The short term memory has a very limited capacity and duration. It reflects the current attention of the reader. The cognitive processing in the long term memory has an unlimited capacity and duration but is very slow and sequential. Thus, the effective use of the retinal variables helps reducing the cognitive load of the human readers by shifting some of the cognitive work to the visual system.

The massive use of schemes or diagrams in software engineering (95 % of practitioners [21]) is then understandable regarding the added value of the visual representations. Of course, the need of effectiveness when creating this kind of systems has been and remains a real academic and industrial problematic (see Diagrams and VL/HCC conferences or the Journal of Visual Languages and Computing).

To enable object visual modeling tool interoperability, the Unified Modeling Language (UML) standard has taken inspiration from existing works concerning software modeling and has formed during the two last decades a good compromise between human readability and interoperability. UML became the de-facto visual modeling language for visualizing, specifying, constructing and documenting software intensive systems. For that reason, UML is in the focus of this thesis. Practitioners have noted several important benefits in using UML including: having a shareable common representation of the system to be built [13], the ability to mitigate the growing complexity of software by working at higher levels of abstraction and having more effective communication [18]. From a practitioner's point of view, communication seems to be the most common benefit expected from modeling [18] and is ranked first among the important attributes of software models [21]. In addition, UML diagrams are frequently used in support of maintenance tasks, to assist in navigating and understanding the system [18].

1.2 Problem statement

Concerning the areas of improvement in UML, we note the possible difficulty to describe contextual information (essentially relies on the saving of information function). Examples of contextual information can be the reasons why a diagram was written, its most important or central element(s) or the reason of this significance. More specifically, UML does not specify effective ways to visually express extra-information on a diagram. These extra-information is not limited to contextual ones and can also be the will to highlight the recurring use of a pattern (e.g., Model, View and Controller).

Furthermore, surveys about UML use in practice showed its unsuitability, sometimes, as a communication vehicle of a project (communication function) [21]. This is mainly due to differences of stakeholders levels of experience in UML. They might be familiar with it (e.g., software developer) or not (e.g., clients).

Let us illustrate these facts via an example of a class diagram. It is shown in Figure 1.2 below. It describes the structure of an e-learning platform. Three types of users can log in to the platform (i.e., Trainer, Trainee and the Administrator). They can register to a training session, do the activities and use the means of communication which are available (i.e., Video-conference, Messages, Chat and forums).

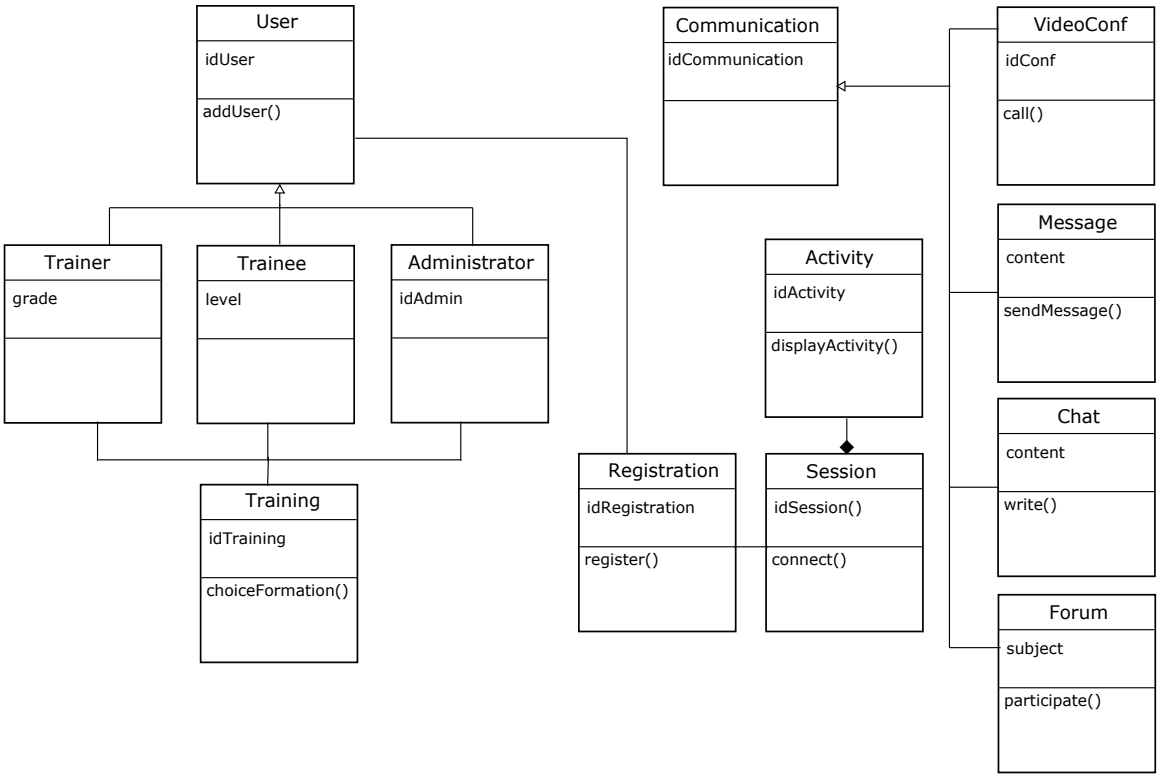

Figure 1.2: Class diagram of an e-learning platform

We assume that a project manager uses the class diagram to discuss the progress of the development of the e-learning platform with the client. For that, he can use the diagram in Figure 1.2 without modifications. But, he adapts his speech to describe the actual state of progress. He might say: *"All the classes concerning the users were implemented, but the administration of the platform part is not yet implemented. We started implementing the video-conference mean of communication. The chat, the messages and the forum options are not yet implemented. The registration to a particular session is already implemented and we begun implementing the activities management part"*. In such case, the client might not be able to keep track of the discussion and to have an overview about the progress of the project. He might wonder: *What is the percentage of the project's progress? What remains to do? Can we estimate the delivery time?*. The project manager can also use the profiling mechanism [4] and annotate each class by a stereotype named by the corresponding state of progress, as illustrated in Figure 1.3 below. He can also add text within UML comments. Such usages are recommended by [34], however they might not represent the best solutions. In fact, textual representations processing is sequential and serial compared to visual

representations processing which is parallel as previously explained.

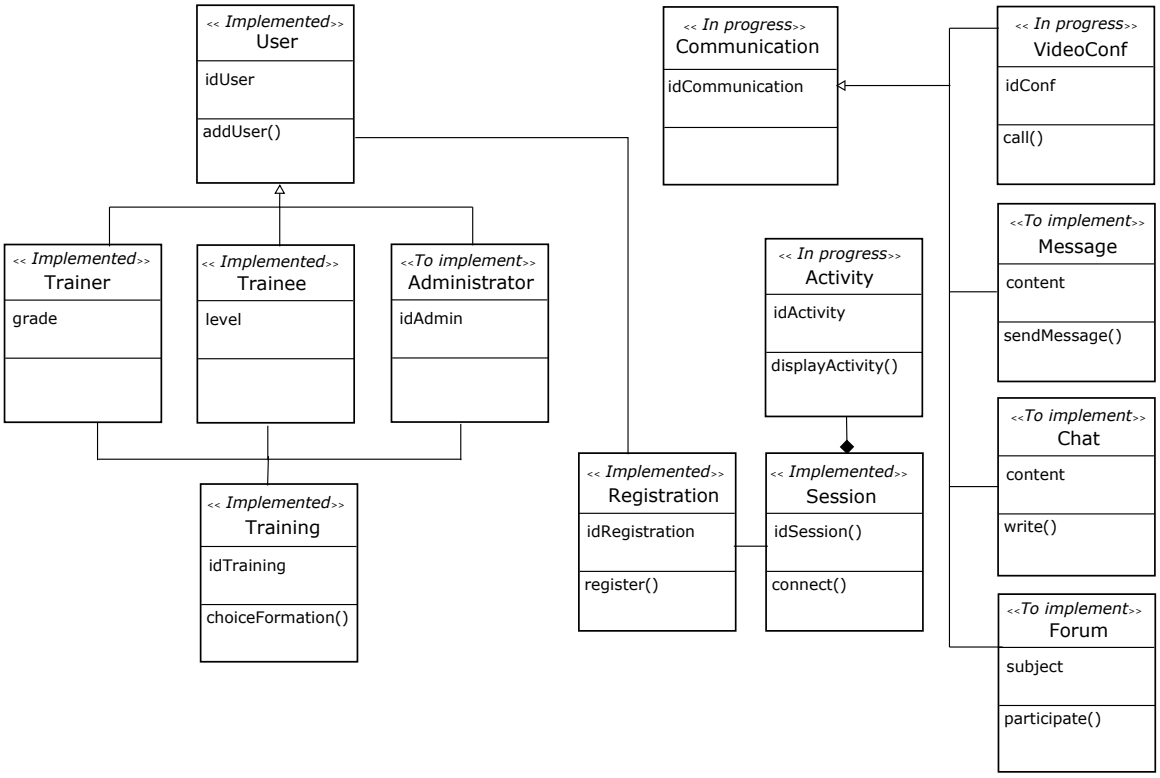

Figure 1.3: Class diagram of an e-learning platform: distribution of tasks using stereotypes

A legitimate question arises at this level: how to improve or upgrade the effectiveness of UML to address these facts?

In software engineering, there are two major conceptual frameworks that deal with the cognitive effectiveness [2] of visual notations in software engineering (i.e., including UML): the Cognitive Dimensions (CDs) framework [25] and the Physics of Notations (PoNs) framework [34]. The CDs framework is an evaluation technique for programming environments [3]. It captures a discussion support about cognitively relevant aspects of structure of programming environments. However, it is considered as only an evaluation framework, providing guidelines to enhance the cognitive effectiveness in each CD is out of its scope. The PoNs framework consists of a scientific framework to create new cognitively effective visual notations and to evaluate existing ones. It explores the power of the visual system to improve the cognitive effectiveness in some dimensions of the CDs framework. The author of the PoNs has structured a scientific basis for visual notations in software engineering. Nevertheless, the author has synthesized research theories which led to some incompleteness in some principles. Designers of visual notations cannot use them directly [42].

1.3 Research questions

In the context of UML, changing its specification (i.e., its concrete and/or abstract syntax) to address the aforementioned facts does not represent the best and the only

[2]The cognitive effectiveness denotes the speed, ease, and accuracy with which a representation can be processed by the human mind.

[3]Visual notations like UML and associated tools

solution. In fact, if we refer to the CDs framework, we can state for UML that the OMG [4] has precisely and exhaustively specified its primary notation and its semantics. But, it does not specify anything about the secondary notation dimension [25] which refers to the free use and change of the visual variables that are not employed in the UML primary notation (i.e., color, size, brightness, orientation and grain). There is no rule governing the way in which the visual variables of the secondary notation can be used. This leaves a big freedom and a possible mean to make UML more effective in saving, communicating and treating information especially by taking advantage of the visual variables. These latter are particularly powerful in reducing the cognitive work required to read and understand the rational/information conveyed by the diagrams and to navigate between them [34]. In that context, the following question arises [4]:

RQ: How to improve the effectiveness of UML diagrams via an advanced usage of their secondary notation visual variables?

1.4 Contributions

Addressing this question represents the main research axis of this thesis. The present work is considered as a continuity of T.R. Green work regarding the CDs framework and Moody's work regarding the PoNs framework. The main characteristic of this thesis is to explore the rules defined by the Semiology of Graphics (SoG) [10] in the UML context/ work setting.

The SoG is one of the main references in cartography. It has been for a long time adopted in the visualization of information works, in software visualization [30] and in HCI, particularly in the work of Yuri Engelhart with one of the authors of the cognitive dimensions [11]. In software engineering, it has been referenced by Moody in the PoNs framework and in the works of Conversy [14].

The exploration of the visual variables in UML requires understanding of; (i) Details about the situations of the use of UML in practice. (ii) Details about the actual state of practice in using (or not) the visual variables in UML. If numerous empirical studies treat the first scope by investigating the use of UML in practice, less researches investigate the use of the visual variables. They mainly focus on the position visual variable to find effective layouts [45] [40] [37] with sometimes studies about colors [46]. In order to fill this gap, we contribute an exploratory empirical study which consists of both qualitative study via in-depth semi-structured interviews with practitioners of UML and quantitative study via the analysis of the use of the visual variables in + 3500 UML diagrams. Among other results, both studies showed out a recurrent non effective use of the visual variables in UML (e.g., no keys). They also revealed an existing need to effectiveness in the use of the visual variables in UML including: the need to the subtlety of the visual variations and the importance of the usability of modeling tools in terms of rapidity, effectiveness and dynamism. This contribution has been published in the *Human Factors in Modeling workshop* at the MODELS conference in 2017 [20].

In order to improve the effectiveness of UML in the situations of its use in practice (i.e., captured in the previous empirical study), we present two additional contributions in this thesis. First, we define a scientific framework describing guidelines for the

[4] This is the general research question of this thesis. It is revisited and refined later on page 57.

visual enrichment of UML diagrams based on the SoG. It begins with synthesizing the rules related to each visual variable from the SoG. At that stage, we observed that the SoG generally deals with elementary marks (e.g., only one geometric or symbolic shape) and not with composed ones which are massively used in UML. So, we proceed to a kind of empirical classification of the elementary UML graphic components. This aims at drawing an exhaustive list of them and to specify the way in which they are bound together. This allows us to carefully adapt the SoG principles to UML. The result is a list of sixty-one modeling guidelines in order to effectively exploit the UML secondary notation except layouts. In fact, although the position visual variable (i.e., layout) is under modeler's control, it is not treated by SoG-UML. We deem that there exist many research works in the literature that deal with layouts like [45] [40] [37]. In addition, animations are not in the scope of this thesis. The guidelines of SoG-UML concern the effective implementation of each retinal variable in the UML graphic components which are contained in a white background. They concern the border, text, background, ports and icons. Then, the guidelines treat the possible containment of UML graphic components having different categories of the retinal variables (e.g., a green class contained in a dark package). Finally, SoG-UML treats the combination of retinal variables on UML nodes. It relies on the design and action theory[5] [26].

To validate the guidelines of SoG-UML, we contribute a validated design methodology of an empirical study having an experiment as strategy of inquiry. The design methodology has been executed with the attendees of the Human Factors in Modeling workshop in 2016 and published in [19]. But, it failed because of problems that we explain in Chapter 5. We list the lessons learned from it to help succeeding the next experiments in the field of this thesis. Then, we detail the experiment that we have conducted and which succeeded to give statistically significant results. It involved ninety-five participants. It shows that the effective use of color helped the participants to find the correct answer in a relatively short period of time compared to the use of text via stereotypes. It validates three guidelines of SoG-UML. The design methodology of this experiment serves to validate the other guidelines of SoG-UML which becomes then a testable theory [26].

The next contribution is a proof of concept tool integrated in the Papyrus environment [24]. First, it consists of an implementation of the classical mechanism of graphical layers found in existing image editors (e.g., gimp) named FlipLayers. The retinal variables and the layers mechanism have been coupled together which aims at enhancing the communication value of UML diagrams. Controlling the visibility of a group of UML nodes together allows managing the complexity [6] of big UML diagrams [34]. It also allows a better cognitive integration[7] of the different views of an UML model. This contribution has been published in the MODELS main conference in 2015 [8]. Second, we provide a prototype of an interactive keys to relatively rapidly decorate UML diagrams using the retinal variables. The interactive keys aims at satisfying some of the recommendations which have been mentioned by the UML practitioners in our previous empirical study.

[5] A design and action theory is meant to give explicit prescriptions for constructing an artifact

[6] The ability of a visual notation to represent information without overloading the human mind.

[7] the cognitive demands on the reader to mentally integrate information from different diagrams and keep track of where they are.

1.5 Outline of the thesis

This thesis is structured as follows:

Chapter 2: The visual variables in the practice of UML presents how software designers use the visual variables in their daily practice of UML. For that, we refer to the results of the empirical study that we have conducted as a first contribution of this thesis [20].

Chapter 3: State of the art proves how the CDs framework represents just an evaluation framework of programming environments. Then, it illustrates how the PoNs Framework principles are incomplete and cannot be directly used by visual notations designers. In a second part, as we propose to refer to the SoG to enhance the cognitive effectiveness of UML, we present a first refinement layer of the SoG in the context of UML. For that, the basic theories of the SoG are exposed with a parallel positioning in regards to UML. It shows that the SoG does not deal with complex graphic elements like UML nodes (e.g., shapes that might contain text, headings, compartments, ports and icons).

Chapter 4: SoG-UML: Semiological guidelines for the visual enrichment of UML diagrams defines the framework which structures the effective use of each retinal variable in UML, by managing the graphic complexity of the UML concrete syntax. The main objective of SoG-UML is to enhance the cognitive effectiveness of UML diagrams by taking advantage of the retinal variables.

Chapter 5: Evaluation establishes a validated design methodology of experiments in order to assess the effective use of the retinal variables in UML. It makes of SoG-UML a testable theory. For that, it presents the lessons learned from a conducted but failed experiment [19]. Then, it defines the design methodology of an experiment which has been executed and succeeded to give statistically significant results.

Chapter 6: Implementations presents how the SoG-UML guidelines might be integrated to modeling tools. For that, it describes the prototypes of tools which have been developed in the Papyrus environment [8].

Chapter 7: Conclusion and perspectives summarizes the contributions of the thesis and exposes its perspectives.

Chapter 2

The visual variables in the practice of UML

Contents

Studying the utility of the visual variables in UML requires a deep exploration of the actual state of practice of both UML and the visual variables. On one hand, there is a huge effort in the literature investigating UML in practice. For example, [13][21][21][16] mainly address the purposes, the costs and benefits and the ways of using UML in practice. But, they do not mention enough details about the situations when practitioners need to visualize information and the kinds of information needed within each situation. On the other hand, few works concentrate on the use of the visual variables in UML. The majority of the existing researches focus on finding the effective layouts (i.e., the position visual variable) based on diagram comprehension and user preferences [45] [40] [37]. Some studies focus on colors like [9] and [46]. All of these works are controlled experiments so they do not report on the practices of UML users and their opinions about the visual variables. There is no qualitative research in this area. We conducted then an empirical study [20] in order to fill this gap. It is described in this Chapter.

2.1 Design methodology

The purpose of the study is to create more and better understanding about the situations of the UML use in practice. A situation refers to the activities, the stakeholders who are involved in each activity, the information that they need to visualize, the practices of UML users in employing UML and the purposes of such usage. In the captured situations, the study aims at discovering the need for the visual variables in practice. If such need exists, we want to gain a great understanding about the kinds of visual annotation that UML practitioners perform, the purposes and the ways to do so. As a triangulation method, we analyzed the use of the visual variables in + 3500 diagrams related to open source projects in ub [27][3]. The theoretical perspective of the study is to help us exploring the benefits of the visual system as a mean of resolving problems that the study might reveal. Obtained results can also help us studying the usefulness of the visual variables in enhancing the effectiveness of UML in the captured situations. Finally, they might help tool vendors enhancing the usability of their tools by making more ergonomic visual automation. The study takes a deliberately broad interpretation of results from both methods, as it is meant to be exploratory.

2.1.1 Qualitative and quantitative methods

We used a selective range of research techniques to gather data for our study. We used both qualitative study via in-depth semi structured interviews and quantitative study via the analysis of UML models related to open source projects in GitHub [3]. Such use of a variety of types of data helps us ensure a better coverage and a greater understanding about our following two research questions:

RQ1

What are the situations when practitioners use UML, particularly which information do they need to visualize in each situation?

RQ2

How and when do the practitioners use the visual variables in the previous situations?

The qualitative in-depth interviews with eight experts and practitioners of UML helps us to gain understanding about the use of UML and the use of the visual variables

in practice. They allow us to understand the relationships between both kinds of use. More precisely, they let us find out the information that the practitioners need to visualize and the ways of using the visual variables (if so) to make them more visible. The analysis of the UML models related to open source projects provides us with quantitative data, particularly, about the use of the visual variables. It mainly answers the second research question by allowing us to draw conclusions about the amount of the visual variables use in a sample of + 3500 UML diagrams. This also enables us to judge the effectiveness (or not) of such usages based on existing theories, the SoG in this thesis [10]. This study is conducted in order to help us to achieve our goal in exploring the high performances of the visual variables in UML.

2.1.2 Interpretation of results

We need to be particularly careful about how we analyze the results of our study and the conclusions that will be drawn. In fact, the first intent of this work is to create better understanding about the use of UML and the use of the visual variables in practice. It is not meant to verify hypothesis or generalize findings. It mainly serves as an exploratory study to help ongoing researches around UML.

The analysis of our interview data has been carried out using the 'grounded theory' which is defined as a qualitative strategy in which the researcher derives a general abstract theory of a process action or interaction grounded in the views of participants in a study [41]. For that, we began by manually transcribing the interviews from audio to textual form. We read throughout the data and identified themes and descriptions. We tried to interrelate them using the grounded theory approach then we interpreted the results.

The analysis of the UML models related to open source projects involved some basic enumerations and simple statistical calculations to get overall sense about the use of the visual variables in UML. The major effort was spent on the manual classification of the different diagrams based on the different usages (or not) of each visual variable. That helped us reporting on the state of practices of UML modelers in using the visual variables. You can find a replication package of the study in [6].

2.1.3 Data collection procedures

Qualitative empirical study: In depth semi-structured interviews

We conducted a series of semi-structured in-depth interviews with eight participants (six from industry and two researchers). Seven interviews have been carried out by phone and one was a face to face interview. As the first intent of the present work is to understand in depth the use of UML in practice, we were particularly interested by practitioners of UML. They come from a variety of backgrounds and with a range of expertise in UML. The interviews lasted approximately 30-60 minutes and began with a brief announcement of the goal of the study. We also introduced the fact that interviews would be anonymous and asked permission to record them. Then, we asked participants about their current position and level of experience with modeling using UML. We continued with questions about the situations of their use of UML to answer our first research question. That included the purposes of using UML, the activities done with UML diagrams, the employed diagrams, the reasons of using a particular

diagram, the sought information and the ways of using UML in a project from the beginning until the final steps. Then, we asked questions about their current use (or not) of the visual variables in practice. That concerned the identification of the utility of the visual variables in practice, the most used visual variables and the ways of their use. As for any semi-structured interviews, we have identified a number of topics that had to be covered in each interview. We also strongly encouraged participants to explain the details of their claims by pointing out that the minor detail is very important for our study. All the interviews have been conducted in a discussion mode where the interviewer followed the logic and the reasoning of the participants. Finally, the interviews were recorded and transcribed with the permission of the participants.

Quantitative empirical study: Analysis of real UML models

We manually analyzed how the visual variables were employed in + 3500 UML models related to open source projects in GitHub [3] [27]. Most of these diagrams are class diagrams, exactly 3328 class diagrams, 392 are sequence diagrams (The models repository is already biased towards structural (class) models [3]). That aims at gathering quantitative data that might reinforce the interviews results. To that end, we first began by identifying the visual variables that we would study, notably: the size, brightness, color, texture and orientation. Then, we manually classified the UML diagrams based on the visual variables they contain (In case of two or more visual variables, we created a new dedicated folder). For each visual variable (i.e., for each folder), we classified the diagrams according to the kind of its implementation to UML elements. In fact, we observed that each visual variable might be differently applied to them: on the border, text, background, edges, heads and/or compartments. We also differentiated significant visual variables variations and non significant ones. A visual variation is considered as significant if there are different categories of this latter in a same diagram (e.g., blue, green, red are different categories of the color visual variable). They mean that authors of the corresponding diagrams wanted to express an information using a particular visual variation. Such kind of variations is very important for our study. We will further concentrate on their analysis to understand in depth their use and answer our second research question. Non significant variation refers to the use of a single category of a visual variable (e.g., all the classes are yellow).

2.2 The qualitative study

We have identified interviewees by searching practitioners who regularly use UML. We asked *all* our contacts in order to identify industrial practitioners who might be willing to be interviewed. We have first done an announcement on mailing lists containing potential practitioners of UML: Papyrus tool developers and users community. We have received two answers to that announcement. The first one has been discarded because the corresponding profile did not match with our target population. The second one was retained because he had the adequate target profile: practitioner and UML expert. Then, we sent direct mails to industrial experienced practitioners in the MDE community. We asked them to participate in our study or ask other potential people who might be interested and interesting for our study. We contacted eleven people, among them six accepted our request, one person suggested another one who he deemed more interesting for our study and who was retained, two people did not answer to our mails and finally two indirect contacts did not accept to participate in our study because

they were not experts and practitioners of UML. Below, it is an extract of an e-mail sent to the participants:
"....we are conducting an empirical study which aims at understanding in depth the use of UML in practice. As a result, adequate solutions can be provided to enhance the effectiveness of UML in real situations. In that context, we are searching for UML practitioners for a 30 minutes to one hour interview. If you have some time to discuss, please send me an e-mail to: yosser.elahmar@cea.fr... I would like to emphasize that the collected data will be anonymous...."
In total, we carried out eight interviews with eight participants who are all experts and practitioners of UML. Roles of the interviewees range from[1] the requirement manager (1), software architect (1), software designer (2), consultants (1), software engineers (1) and researchers (2). They work on different domains: transportation, aerospace engineering and defense, avionics, telecommunication, E-commerce, insurance and banking. Five hours and a half of interviews were recorded and manually transcribed.

2.2.1 Analysis

Situations of the use of UML in practice

Purposes of the use of UML The results of the qualitative study revealed that communication is ranked first among the usages of UML in practice. The eight participants have confirmed their use of UML diagrams as a communication vehicle. Communications might be held internally within the project teams or with costumers. This finding is also confirmed by the empirical studies that exist in this field like [21][18]. The next paragraph further focuses on our results about UML use in communications. The second purpose of using UML is code generation. This finding is contradictory with previous empirical studies [16] where code generation generally appears in the last ranges. However, that seems logic in our case because most of our interviewees are MDE approaches practitioners. They use models from early design steps until maintenance tasks. The third purpose of using UML is to draw the participant's own understanding, in an informal way where UML diagrams are considered as a "map of the system". That might be done using a pen and paper or on a white board. In such kind of use, participants do not care about the conformity of their diagrams to the UML standard. Their goal mainly concerns the comprehension of the system to be built and its conformity to the clients needs. Finally, UML diagrams are less employed for model execution and model analysis.

UML and communications We asked our practitioners about their practices of using UML diagrams for communications. We distinguished two types of audiences: people who are familiar with UML (e.g., technical team) and non-familiar with UML (e.g. customers). We found out that none of our practitioners modify (i.e., contextualize) their diagrams for communications with people who are familiar with UML. They argue that all the stakeholders already know and understand the language. However, when it is about discussing with customers, they react differently (Figure 2.1). Most of the practitioners do not modify their diagrams but try to adapt their speech to the audience. Below, you will find two claims from our practitioners:
"... We kind of read the diagram to them then we say our interpretation and they just hear what we say and they agree or not with that..." (Transcript 3)

[1]The number in brackets refers to the number of participants having the corresponding role.

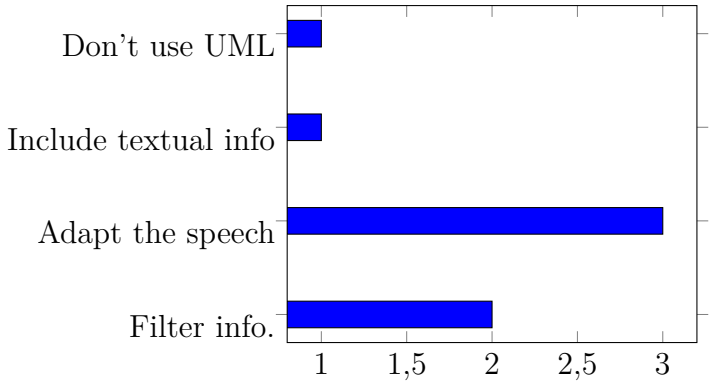

Figure 2.1: Communications with non-familiars with UML

"I didn't ask him to learn all of UML but like for the class diagram I would explain the class you know what the class is, the attributes and relationships that takes only a few minutes and then... the subject matter he is really familiar when he sees that these boxes as you know class called solution or column or pump and types of things that are easier to work with..." (Transcript 4)
Other interviewees would prefer to filter some information from their diagrams to keep only those interesting for their communications. To that end, they omit technical details that don't really matter to their customers. They try to keep diagrams simple to better communicate.
"...we actually try to simplify as much as possible in our ... UML model because they aren't UML experts so we try to filter out all... We try not to overload our diagrams with labels everywhere that non UML experts will not understand" (Transcript 7)
Finally, one practitioner prefers not to use UML when discussing with people who are non familiar with UML.
Generally, all our practitioners were aware of the unsuitability of UML for all types of communications. They try to find different manners to facilitate such use. Rare of the practitioners has mentioned the recurrent use of the visual variables to adapt the diagrams to communications. This fact is mainly due to problems with tools (see Section 6).

Used UML diagrams Interviews showed that class diagrams and sequence diagrams are the most used in practice (Figure 2.2 below). Different reasons are given to justify the choice of such particular diagrams. A software engineer argues that the class diagram is the most expressive notation in UML for modeling data. A software designer uses the class diagram to have a design of the database. A software architect pointed out that the class diagram is used to divide the work among the different teams which are involved in a same project. Class diagrams are also employed to draw the business entities of the systems and to represent the functional relationships between these latter. Concerning the sequence diagrams, they are mainly used to define the interaction between the classes and interactions between users and the solution (i,e., the common definition of a sequence diagram). Sequence diagrams are also used in the definition of the white box part of a solution and to realize specific use cases. The use case diagrams and the state machines diagrams are ranked second among the most used UML diagrams in practice. The purpose of creating use case diagrams is to enumerate the functions to develop and to specify actors and the interactions between them. Eventually, this refers to the definition itself of a use case diagram. The

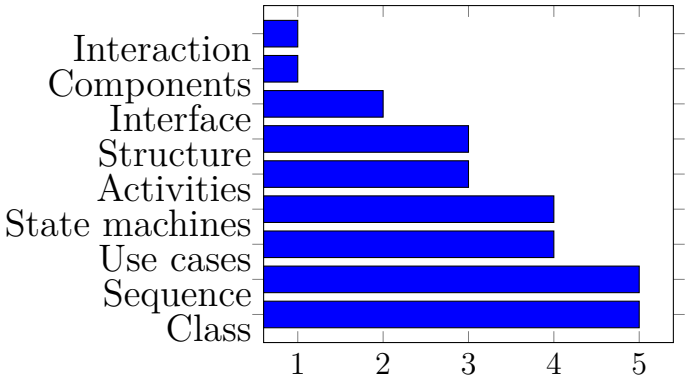

Figure 2.2: UML diagrams used in practice

requirement manager justifies his use of the use case diagrams by the fact that such use is recommended by the safety requirement standard. Use cases are also used to drive the software engineer thinking and they will be part of the documentation. State machines are mostly used to design the behavior of the systems to be built or as an executable model. Activity and structure diagrams are the fourth most used diagrams by our practitioners. Activity diagrams are mostly seen as an elaboration of the use cases and a representation of the systems features. They are also used for the business process modeling and as a communication vehicle with customers. Then, on the last position, come the interface, component and interaction diagrams. These findings are coherent with previous empirical studies led in this area [16][18].

Pattern of UML use in practice We asked the interviewees to describe in detail their practices in using UML to build a system or a project. We analyzed the answers to this question and were able to identify a pattern of the use of UML by our interviewees. All of our practitioners begin with gathering the requirements from the customer. This might be done in a textual form or via a modeling session.
"The three people working on the project for example, we interview users who want the system and we understand from them what the requirements are, then we translate these requirements. It is like we have a modeling session, we sit with them the three of us and we interview that, what do you imagine blablabla. And then we capture the use cases and we start populating a use case diagram..." (Transcript 3)
At this level, the models serve as a support of communication with the customer and within the technical team members. This step allows our participants to draw the big picture of the systems to be built. One interviewee mentioned the advantages of representing the system in a visual form instead of text.
"Drawing the system instead of writing is a good tool to communicate and share mind viewpoint. The vision goes more quickly, we can decide more quickly about the architecture, the architecting stuff." (Transcript 6)
Then, participants move to an understanding session where they review and check the requirements of the customer to ensure that they fit with their customer's needs.
"We want to represent the system as it is and we want to understand the needs may be to understand the way to go to the system to be. So we used different diagrams offered, provided by UML to draw the big picture of the – context to deeply understand what is the need." (Transcript 6)
To that end, they might need to go back to the customer and review the requirements in another modeling session. Once ensured that their models match well with the re-

quirements of the customer, they split the work among the people who are involved in that project. To that end, UML might be employed as a discussion vehicle via the class and the use case diagrams. Finally, each participant continue to use UML for her/his particular needs: model simulation and execution where the models represent the code. They might generate code from them or continue coding the system and keep the created models in the documentation. In all cases, the models will populate the documentation which describes each project or system. To these ends, most of our practitioners use a modeling tool in their practice. One interviewee pointed out that the use of a modeling tool depends on his needs. If it is about gaining his own understanding of the system, he settles for a pen and paper. Otherwise, if it is about automation, he does use a modeling tool.

Searched information We asked our practitioners about information they need to visualize in practice. We distinguish two types of information. The first one refers to the semantic information (i.e., what is modeled in a diagram). Below are examples of semantic information mentioned by our interviewees:

- **Input and output statements for the requirements**: the interviewee who mentioned this semantic information is a requirement manager. In her/his practice, she/he begins first by gathering the requirements from the customer. Second, she/he reviews the requirements and draws the use case and the state machine diagrams to build her/his own understanding. Then, she/he reviews the requirements. For that, she/he needs to check if the input requirements that are given by the customer correspond to the output requirements that are represented in the use case and the state machine diagrams.
- **The communication between the objects in a sequence diagram**: the UML practitioner visually navigates in her/his sequence diagram to see the messages that are exchanged between the objects of her/his system. Such visualization helps her/him understanding the logic of those communications.
- **The concerns of each subsystem**: the UML practitioner wants to see the functionalities of each group of elements in her/his UML diagrams. For example, a subsystem of an e-learning platform concern *the users management* and another one concerns *the database management*.
- **The calls of functions between the diagrams of a model**: the UML practitioner needs to visualize the relationships between different views of an UML model. For example, she/he navigates between an UML class and its behaviour in the state machine diagram.
- **The interactions of a practitioner's own system with the other subsystems**: the UML practitioner needs to see the communication of his subsystem with the other practitioners subsystems. Such visualization helps her/him updating her/his subsystem when a new interaction to it is created or updated.
- **References for specific signals or events in the model**: the UML practitioner needs to find the elements that reference other ones in her/his UML diagrams.

Second, we find what we call extra-semantic information. It consists in non-semantic information but which can be extracted from an UML model. Examples of extra-semantic information are:

- **Level of implementation of the classes**: the UML practitioner needs to see the classes that are already implemented, those that are in progress and finally the classes that are not yet implemented.
- **Bugs in the model in the case of model execution**: before executing her/his UML diagram, the UML practitioner checks his diagram through model validation. As a result, she/he can visualize the existing bugs (e.g., with red markers) and fix them.

We observe that practitioners need to visualize information on their diagrams. Before going to the documentation, UML diagrams are subjects of many visualizations where practitioners need to search for important information to accomplish their tasks. If we link this finding to the previous results about the purposes of using UML in practice, most of the searched information belong to the "drawing of understanding" purpose:

The practitioners visually navigate in their diagrams to find accurate information to build the mental map of their systems or projects.

The visual variables in practice

This subsection analyzes the answers of the practitioners about the second research question: the use of the visual variables in practice. We first begun by asking our interviewees about details of their use (or not) of color in practice. Then, we asked them about their alternative means in case they have color-blind people in their teams. It allowed us to introduce our question about the use of the other visual variables (i.e., size, brightness, texture/grain, orientation). We presented a brief presentation of each visual variable if necessary.

Color We asked our practitioners about their need for colors in practice and about examples of information they needed to highlight using them. Again, we distinguish two types of information: semantic information and extra-semantic information. Only two semantic information were mentioned by one single practitioner:

- Important features like inheritance
- Elements which have the same semantic like interfaces

Most of the interviewees used color to highlight extra-semantic information. The progress of the implementation of classes has been mentioned by three practitioners. They want to visualize the progress of the development of their classes directly on the diagrams. Examples of extra-information are mentioned below:

- Role in the design (criticality, parts of patterns (especially MVC), parts of layers, levels of security).
- Status in development (progress in implementation, testing, execution).
- Distribution of tasks between the stakeholders (ownership of each class).

Besides, one practitioner mentioned that color must not be used to highlight semantic information. He argues that the diagram should be understood without coloring because colors might disappear in case of black and white printing.
"We have discussed and said that we should avoid coloring. At least if the colors have

a specific semantic I mean you should be able to understand the diagram without the colors we can't put any semantic meaning into the colors because if you lose the colors when you print into black and white printers I mean it is pretty fundamental to still have the same semantic of the diagram". (Transcript 5)
Furthermore, we observe that most of the examples of the highlighted information using color are "selective" information: Practitioners want to highlight UML nodes belonging to a same group (e.g., MVC elements, elements that have the same semantics) together. They may occasionally use colors for "ordered" information (e.g., progress of implementation, important features).

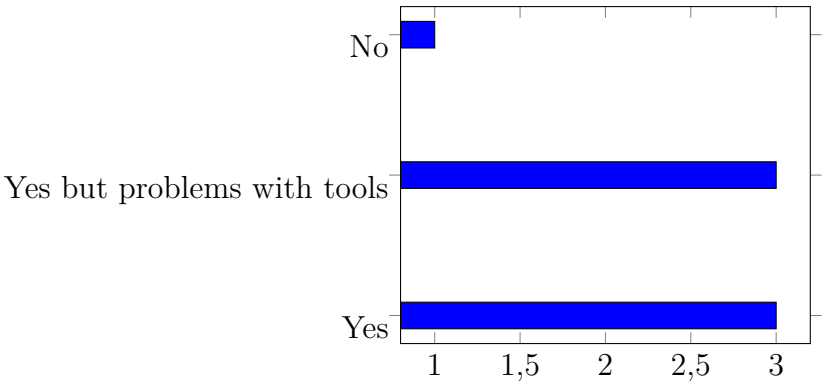

Figure 2.3: Utility of colors in practice

Utility of colors in practice We asked our practitioners if the previously mentioned use of color has been helpful. We found out that most of them agree on the added value of colors and that their use was helpful in practice. Figure 2.3 details the answers of the participants [2]. Three interviewees totally agree on the utility of colors in practice. The same number of interviewees confirm that colors are helpful in practice but there are problems with modeling tools that hamper such use. Besides, they express their need for an automatic and efficient tool and propose some recommendations that will be discussed in Section 6. One interviewee stresses on the fact that colors are helpful but only for communications.

The use of color In the case of the use of colors, we wanted to understand how practitioners do chose them. We found out that only two practitioners use some internal conventions of their companies. Below are examples of conventions used within two different companies:
"Non-tested functions: Blue; safety functions: yellow..." (Transcript 1)
"To communicate the green means we have it, yellow means in progress, red means we –" (Transcript 3)
The majority of interviewees do not have internal conventions, they follow their own tastes.
"In my domain which are in general embedded systems we can use blue for that software functional related, I use orange for everything that software platform related to framework system, drivers, etc and red for everything that is material, hardware related." (Transcript 7)
"I avoid red because red means mistake and green is nice because it means correct."

[2] With one missing answer.

(Transcript 1)
One practitioner says that they have internal conventions but they are used in an ad-hoc manner.
"Unfortunately, this (internal reference documents) is used in an ad-hoc manner. We have just documents to follow but no body follows them in a formal manner." (Transcript 6)
Then, we wanted to know if practitioners add legends/keys when they use colors. We found out that the majority of practitioners do add legends or would like to do so: two practitioners confirm that if they use color, they add keys. Two other practitioners would like to add keys but there are limitations in the used modeling tools (Figure 2.4).

At this level, we observe that most of the practitioners neither follow internal con-

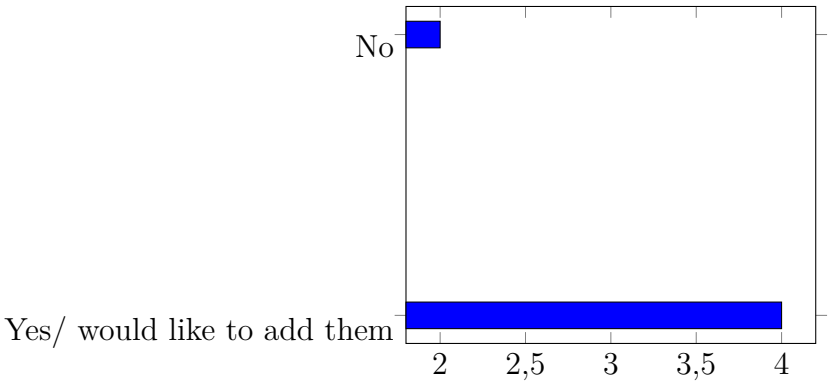

Figure 2.4: The need for keys in practice

ventions nor add keys when they use color. Such behavior is non effective because keys are primordial if at least one visual variation does exist [10]. That might create ambiguities to understand the diagrams in question (e.g., for the author himself after a long time or for another team member who might need to read it while maintenance tasks).

The other visual variables We asked an open ended question about the utility of the other visual variables (i.e.; size, brightness, texture/grain and orientation) in practice. The majority of our practitioners confirm that the use of the visual variables might help using UML in practice (Figure 2.5). In parallel, they stressed on the effec-

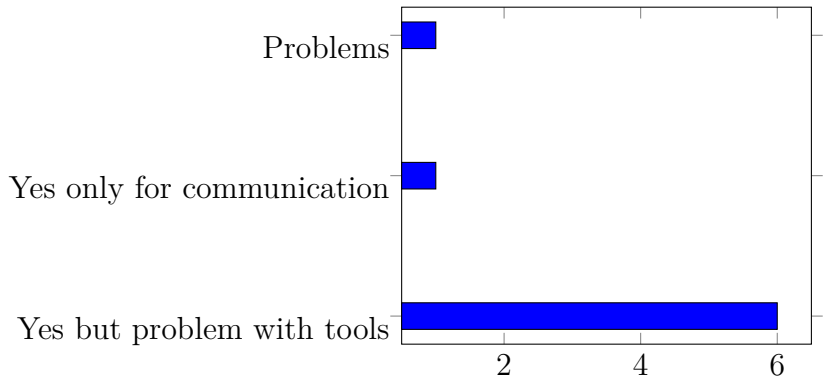

Figure 2.5: The need for the visual variables in practice

tiveness and usability of the employed tools for that purpose. The utility of the visual

variables directly depends on the efficiency and usability of the tools. One interviewee argues that these visual variables might be helpful only for communications. If it is about understanding or using his own diagrams (i.e.; that he creates), he will not use them.
"If I have to model a function, I AM the designer, I am modeling this function, so I don't see how I should use visual annotations." (Transcript 1)
Another interviewee thinks that the use of the visual variables might impact the readability of the diagrams: The size variation might make big diagrams less readable. Texture might also create problems of readability and printing issues.
"Size: No because most of the time, the models are so complex. So having classes bigger than others make the diagram less readable." (Transcript 8)
"Texture: The diagrams are printed and stuck in the wall so using texture... to me it is making the model less readable... it could be more beautiful for business people. For technical people I don't think it will be added value." (Transcript 8)
The use of the visual variables depends also on the size of the working teams. In large organizations, the use the different visual variables might create a mess.
"In smaller teams, probably they are perfectly well where you can align and decide the coloring rules and so on but as long as get a little bit bigger, then going and using different visual variables... just creates a mess." (Transcript 5)
Ongoing researches about the use of the visual variables in UML should take into account these claims and provide effective material (i.e., via theories and convenient tools) to handle the aforementioned problems.

2.3 The quantitative study

We have randomly chosen a sample of +3500 diagrams from the GitHub models repository [3]. The repository was already biased towards structural models. That explains the fact that 90% of our sample were class diagrams. In that context, 22% of the

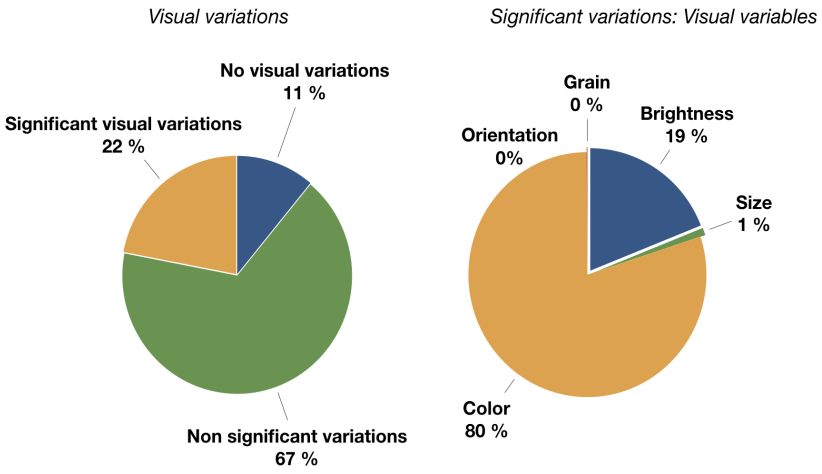

Figure 2.6: Analysis of the visual variations in the models repository.

diagrams of our sample present significant visual variations (Figure 2.6). That means that modelers did need to highlight information and used the visual variables to that end. As depicted in Figure 2.6, color, brightness and size are the three visual variables which are the most used. We found out that only one diagram is using the texture visual variable and that the orientation is never used. 67 % of the diagrams present

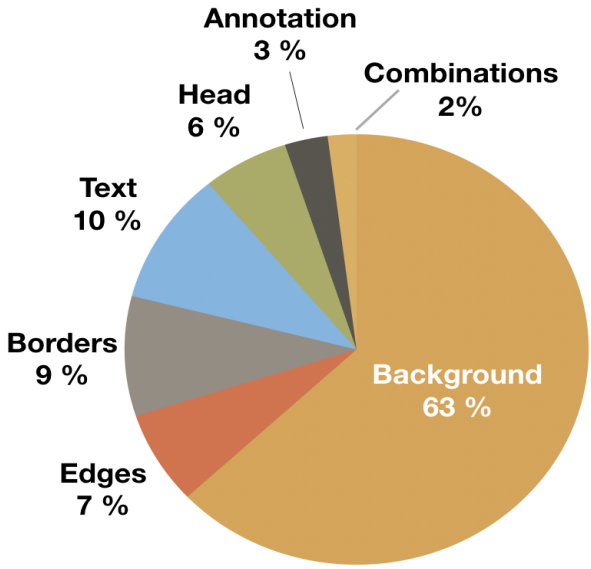

Figure 2.7: Different implementations of color to UML elements

non significant variations (Figure 2.6). Such non-significant variations refer to the default configurations of the used modeling tools (e.g., by default, all the classes might be yellow, blue, green, gray, etc,.). 11% of the diagrams are purely black and white ones and do not present visual variations.

2.3.1 Analysis

Color

80% of the significant visual variations are expressed by using colors (Figure 2.6). We analyzed details about such use and observed that they are differently applied to UML elements: background, borders, edges, text, heads and compartments. We found out that colors are applied to the background of the UML elements in 63% of the diagrams that present significant color variations (i.e., classes or lifelines) (Figure 2.7). 10% of the diagrams present a color variation of the contained text of an UML element: class name, attributes, methods or even text related to comments. 9% of the diagrams present color variations of the UML element's borders, particularly the borders of UML packages and classes. Finally, we observed that modelers add information in their diagrams using colored text or arrows. We looked further into detail to find out the information that modelers wanted to highlight. However, it was difficult to identify them. This is due to the lack of keys or any information that designates the meaning of each color variation. In fact, only 4% of the diagrams which present a color variation do contain keys or simply meanings of the applied visual variations (Figure 2.8). 14% of these keys are not up-to-date with the corresponding diagram. That might occur because the used tool does not automatically update the keys. This observation is akin to the previously mentioned statements in which practitioners have raised their need to add keys and pointed out the limitations of tools to add these latter. The analysis of the diagrams where keys are available showed that colors were always employed to express selective information. Most of these diagrams present the Model, View and Controller elements as highlighted information. They use the following sets of colors: (pink, yellow and mauve), (green, yellow and mauve), (blue, orange and green) and (yellow, green and red).

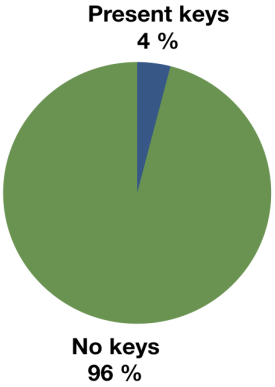

Figure 2.8: The use of keys/legends in UML

Brightness and size

As mentioned above, the brightness is the second used visual variable. As for color, brightness is mostly employed to highlight selective information. Modelers chose different levels of brightness of a particular color or ranges of white and gray. Brightness is always applied to the background of UML elements.
For the size variations, significant ones are mostly applied to text. Modelers change the thickness of the text that they want to emphasize (e.g., attributes, methods or just a part of them). Once, different sizes of the text have been used.

2.4 Discussion of the results

The results of the interviews analysis show that UML diagrams are employed in several situations (e.g., communication, drawing of understanding, analysis) using different diagrams (e.g., classes, activities, state machines). These situations involve many visualization tasks where practitioners need to research information important to accomplish their work. These information might be semantic or extra-semantic ones. The interviews also show that colors are sometimes used in practice. Such use has been recognized as helpful by our practitioners. Concerning the other visual variables (i.e., size, brightness, texture/grain and orientation), practitioners do not actually use them but deem that they might be helpful and useful in practice depending, of course, on the usability of the modeling tools. To reinforce their claims, practitioners mention recommendations about effective ones (i.e., tools). First, they express the need to an automatic tool that updates the visual variations when the information change or evolve. Concerning the extra-semantic information about the progress of the implementation, classes should automatically be updated when a class status moves from in progress status to implemented one. Practitioners have also raised the need to add keys when they use colors. They pointed out that not all the modeling tools present such feature. In that context, they suggest to have interactive keys that enables, for instance, the possible update of the visual variations in the UML diagram in the keys and vice versa. They also recommended the possibility to define rules which map the information to be highlighted and the corresponding visual variable. Furthermore, practitioners stress the subtlety of the used visual variations. The visual variables have to be associated with particular meanings. They also stress the necessity to consider large organizations where a big number of people collaborate on the same models: the tool should handle the conflicts that might appear.

As with the interviews, the results of the quantitative analysis of the UML models show that color is the most used visual variable. But, concerning the other visual variables, it shows that brightness and size are also used to highlight information. In the models repository, only 4% of the diagrams present keys and sometimes they are not up-to-date with the corresponding diagrams. The latter finding confirms the interviews results about the need of an automatic tool. Existing theories like [10] prove that keys are mandatory when at least one visual variation does exist in a graphical representation. It helps reading and understanding the meanings of such variations. Indeed, when we tried to analyze the models in the repository [3], we encountered problems to understand the meanings of the visual variations applied by modelers. That might be problematic in practice when modelers want to understand the diagrams which contain some significant visual variations. In addition, we observed that the used visual variables are differently applied to UML elements: border, background, text, etc. In that context, there are absolutely implementations which are more effective than others. Via both research methods, we observed that colors are mostly employed to express selective information. Based on [10], such use is effective. Selectivity is one of the perceptive properties of colors thanks to which the human eye can rapidly select groups of elements having the same color together. However, we also noticed that practitioners use colors to express ordered information (e.g., the progress of the implementation of a project). Such use is non effective [10]. In fact, the human eye cannot order colors but it can spontaneously and rapidly order different levels of brightness (i.e., from dark to bright and vice versa).

2.5 Conclusion

The present empirical study provides understanding of the use of UML and the visual variables in practice. Eight interviews have been carried out with experts and practitioners of UML. In addition, + 3500 UML diagrams were analyzed to discover the employed visual variables and discuss the ways of their usage. Below are synthesized the results revealed by the study:

UML in practice

- Communication is ranked first among the important attributes of UML models (See paragraph *Purposes of the use of UML*, page 13).
- Class diagrams and sequence diagrams are the two diagrams which are the most used by the practitioners (See paragraph *Used UML diagrams*, page 14).
- Practitioners noticed difficulties to use UML diagrams as a support to discuss with people who are not familiar with this standard notation. They react differently to compensate for this (i.e., adapt the speech, filter some technical information, include textual information) (See paragraph *UML and communications*, page 13).
- There is a need to visualize information in UML diagrams: functionalities of the system, source and destination of a message, input and output statements, information related to project management (See paragraph *Searched information*, page 16).

UML and the visual variables

- Most of the practitioners think that the visual variables might be useful in practice (See paragraph *The other visual variables*, page 19).
- Color is the most used visual variable (See paragraph *Color*, page 21).
- A recurrent non-effective use of colors exists (e.g., no keys, colors to express ordered information) (See paragraph *Color*, page 21).
- Different implementations of the visual variables are used: Background, borders, text (See paragraph *Color*, page 21).

What remains to do with the visual variables?

- There is a need to study the effectiveness in the use of the visual variables in practice (See paragraph *Discussion of the results*, page 22).
- Importance of the subtlety of the visual variations (i.e., to associate a meaning to each variation) See paragraph *Discussion of the results*, page 22).
- Importance of the usability of modeling tools: automatic, interactive keys, rules about the effective mapping between the information to be highlighted and the visual variables See paragraph *Discussion of the results*, page 22).

These empirical results have definitively and strongly motivated us to propose the adequate solutions in this thesis. What we were looking for were theories to effectively employ the visual variables in UML. For that, we elaborated a framework called SoG-UML that we present in Chapter 4. Such a theory was missing in the MDE community. This is what we expose in the next Chapter with a complete state of the art. We also implemented prototypes of automatic tools in the Papyrus environment [24] which respect some of the practitioners recommendations that we describe in Chapter 6.

Chapter 3

State of the art

Contents

As this thesis targets the cognitive effectiveness of the visual modeling language UML, this Chapter discusses the main scientific works which deal with it in the whole software engineering field, notably the Cognitive Dimensions (CDs) framework [25] and the Physics of Notations (PoNs) framework [34]. For that, we describe both frameworks and position the work of this thesis in regards to them. We show that the CDs does not provide solutions to lighten the cognitive load of human beings in each dimension. We also show how the PoNs cannot be directly employed to evaluate or create effective visual notations. We conclude that our work complements both of them by exploring the power of the visual system. To that end, we expose the basic theories of the Semiology of Graphics (SoG) along with a parallel positioning in relation to UML.

3.1 Software engineering

This section states the two conceptual frameworks for visual notations which exist in the literature: the CDs and the PoNs frameworks. On one hand, the CDs framework emphasizes the importance of having cognitively effective programming environments [1]. On the other hand, the PoNs framework explores the visual system to build effective visual notations or evaluate existing ones. Strengthening the effectiveness of visual notations represents then a common objective between the work of this thesis and both of the frameworks. We present here the definition, the strengths and weaknesses of both frameworks. Missing well-formalized guidelines about the use of the visual variables is the conclusion of this Section and one of the main motivations of our work.

3.1.1 The Cognitive Dimensions framework

The cognitive dimensions framework is an evaluation technique for programming environments. It captures a discussion support about cognitively relevant aspects of structure of programming environments. These cognitive aspects are called Cognitive Dimensions (CDs). The CD framework serves as a starting point for later researches about programming environments. We will discuss its limitations in giving solutions to reduce the cognitive load which might be revealed by it.

The thirteen cognitive dimensions

The 1980s and 1990s witnessed the emergence of the first Visual Programming Languages VPLs (e.g., Entity Relationship diagrams (ER) in 1976). The same period has been characterized by the advanced researches in psychology of programmers. It is in that context that the CD framework has appeared. Based on researches about the psychology of programmers, the authors consider that programming activity concerns not only coding (or modeling), nor just comprehension. It is as much affected by the programming environment, where programming environments include interactive devices like modeling tools and non-interactive notations like UML. In that context, the authors distinguish between "Psychology of programming" and "HCI of programming". The psychology of programming has to do with the coding or modeling part (i.e., the

[1] We use terms employed by the authors of the cognitive dimensions framework. In the context of our paper, programming activity denotes modeling activity. Programmers can be developers (coders) or UML practitioners. Finally, the programming environment is composed of the modeling language and the modeling tool.

meaning of the code or the model). HCI of programming is about the interactions between programmers and the programming environment. Researchers define four types of interactive activities that programmers perform while programming: search of information (e.g., where is the lifeline named "Client"), reuse (e.g., reuse a part of a UML diagram in another), modification (e.g., update the name of a UML class and propagate that update to other diagrams) and navigation to understand and compare multiple diagrams (e.g., explore the semantically linked UML elements in a model). By studying these activities, the authors became able to define a set of thirteen cognitively relevant aspects of programming environments structure. These aspects are called Cognitive Dimensions (CD) and form the cognitive dimensions framework. The cognitive relevance denotes that undervaluing or non effectiveness in one or more than one aspect of structure (CD) might cause a cognitive load to programmers. Equally, ensuring the effectiveness on them might considerably reduce the cognitive load of programmers. Here is a quick description of each dimension.

Abstraction gradient denotes the levels of abstraction which are provided by a programming language where an abstraction represents a grouping of elements to be treated as one entity. Green et al. define three possible levels of abstractions of programming languages: *abstraction-hating, abstraction-tolerant or abstraction-hungry.* Abstraction hating languages do not provide abstraction facilities or allow a minimum level of abstraction but can be easily learned by novices. An abstraction-tolerant language allows a medium starting level of abstraction and abstraction-hungry languages allow a higher one. But both of them allow new abstractions to be created and are relatively difficult to learn. Programming languages should find the right balance between levels of abstraction and learnability.

Closeness of mapping concerns the mapping between the domain problem and its corresponding code or diagram. The shorter the distance between the concepts of reality and those of the programming language is, the better the cognitive load necessary to understand and infer on the code or diagram is reduced. For example, this dimension serves at evaluating the cognitive load necessary to map an UML activity diagram to the cooking recipe it represents, or an UML state machine diagram to the a pulley system it describes.

Consistency allows to answer the following question: when a person knows some of the language structure, how much of the rest can he/she easily and successfully guess. This dimension allows measuring the degree of difficulty to learn the language. The higher it is, the easier the language is learnable.

Diffuseness/terseness refers to the number of symbols or graphic entities which are necessary to express a meaning using a programming language. On one hand, the bigger the diffuseness is, the more material is needed to scan it (e.g., screens). Also, a big diffuseness causes a smaller proportion that can be held in the working memory and a greater effort to search information through the text. On the other hand, a small terseness (i.e., compactness) makes it difficult to compare two different representations. This dimension allows measuring the degree of the diffuseness/terseness of a programming language. Designers of the language should find the balance between both metrics.

Error proneness allows measuring how well a notation is successful in avoiding

careless mistakes and is accurate in finding the mistake once committed. Following are examples of mechanisms to decrease the error proneness in textual notations: code completion, the declaration of new identifiers, debugging facilities, red markers on the lines of code containing errors. In visual notations, model validation is a way to minimize error proneness.

Hard mental operations indicates if a notation needs hard mental operations to understand it. Conditionals, negations and/or combinations of them require a big cognitive work to reason on such operations. If the human reader resort to another more comprehensible notation (e.g., using a paper and a pencil, touching the screen with fingers to follow the logic), then the notation does imply hard mental operations. Notations should directly transcribe all the possible deductions to lighten the cognitive work of the human readers.

Hidden dependencies measures the degree of visibility of the elements participating in dependencies (i.e., sources and destinations). A classic hidden dependency is an excel cell in a formula which references other cells. Notations should explicitly render the relationships of each element with the others. Such visibility facilitates the search, modifications and deletion of elements. Contrarily, hidden dependencies involve more cognitive work to perform the aforementioned actions.

Premature commitment allows measuring the premature commitment imposed by the notation. Frequently, programmers begin by working in advance mentally or in a paper or leave placeholders to fill later. If the notation imposes in advance many internal dependencies and constraints which restrict the order of doing things, it will require a lot of cognitive work. An example of premature commitments in UML is the necessity to put elements in the right order. A modeler might need to model parts of an activity diagram in a random order. Therefore, she/he should rearrange the layout of the final diagram by putting each part in the right position, which might cause a cognitive load. Modeling tools should provide facilities to simplify such actions.

Progressive evaluation allows verifying if a programming environment enables programmers to frequently evaluate their work even with incomplete versions of it. Code debugging or models simulations and validation are examples of mechanisms allowing progressive evaluations in UML diagrams.

Role expressiveness is related to the expression of the roles of the components of a notation. Roles can be conveyed via the use of meaningful identifiers names, well structured code (e.g., code indentation), the secondary notation (see below), comments or tags. The notation should provide the role of each component in the whole system.

Secondary notation refers to the free use of means which are not formally specified in the formal syntax of the language. This is to add extra-information above and beyond the official semantics. For example, colors are under modeler's control in UML. UML practitioners might need to use them to express, for example, the model, view and controller elements in a class diagram. This dimension serves at evaluating the degree of freedom to use additional means which is given by a notation to programmers.

Viscosity denotes the resistance of a programming environment to local changes.

It serves at evaluating the effort necessary to perform a modification to a model or a program. The programming environment should provide rapid and automatic tools to make modifications more viscous and fluid. Changing the name of an identifier might be rapidly done, but the programming environment should propagate such modification to all the places where the identifier in question has been used. In diagrams, changing the position of a component might take a long time because of the crossing lines which might occur.

Visibility and juxtaposability dimension denotes if all the materials which is necessary to accomplish a task (e.g., parts of diagrams) are accessible without or with a minimal cognitive load: if they can be readily seen, readily accessible to put them visible or readily identified to be accessible (and then visible). For example, an UML practitioner might need to see all the UML nodes which are semantically linked to content of the diagram in front of her/him. The visibility dimension serves at pointing and evaluating the cognitive load required to access these semantically linked nodes.

Advantages of the CDs framework

The CDs framework aims at evaluating the effectiveness of programming environments by assessing the cognitive load in each dimension. In table 3.1 below we have used the CDs framework to evaluate the three most used UML diagrams in practice: Use cases diagram, class diagram and sequence diagram [16] [18]. In this evaluation we consider the diagrams in the Papyrus modeling environment [24]. In each dimension, the framework allows us to state if there is a cognitive load or not via examples of observations that we have made. This evaluation represents our own analysis and it deserves to be completed by other experts. It shows the subjective side of the CDs framework and needs empirical assessment in some dimensions (e.g., the hard mental operations).

Table 3.1: Evaluation of three UML diagrams with the cognitive dimensions framework. ⊕ mark designates a positive point, ⊖ indicates a negative point.

	Use case diagram	Class diagram	Sequence diagram
Abstraction gradient	⊕ Objects, Actors and Edges.	⊕ Objects and Edges.	⊕ Objects, Actors and Events.
Closeness of mapping	⊕ The functionalities of the real system are directly mapped to UML use cases. ⊕ Stick-man is close to the reality. ⊖ All the functionalities are mapped to the same UML object: an UML use case (i.e., an ellipse).	⊕ Objects of the real system and relationships between them are respectively mapped to objects and edges. ⊖ Different objects of the reality are mostly mapped to the same UML entity: an UML class, which is not very close to the reality [2].	⊕ The order of real scenarios is mapped in the sequence diagram by preserving such order. ⊕ Stick-man is close to the reality. ⊖ As for classes, all the objects of the reality are mapped to the same UML entity: an UML lifeline.
Consistency	⊕ If some of the language is learned, the whole diagram can be created.		
Diffuseness	⊖ The diffuseness might be high for big diagrams.		

[2] The use of DSLs can resolve such problem.

	Use case diagram	Class diagram	Sequence diagram
Error proneness	⊕ The use case diagram does not induce to a lot of errors. ⊖ The "include" and "extends" types of edges might be prone to error. They have the same graphical notation but differ by only the corresponding text labels (Conception error).	⊕ Types of the attributes and parameters of the methods are not prone to error. They are all pre-defined in lists in the properties view (Syntactic error). ⊕ Text names are highlighted by a red mark if they contain spaces for example (Syntactic error). ⊖ Choosing the right edges in the palette might be error prone. For example some edges have the same icon (associations, AssociationBranch, ContextLink) (Conception error).	⊖ Might be error prone for superimposed fragments (Conception error).
Hard mental operations	⊖ Extension points are sometimes hard to integrate.	⊕ Does not require hard mental operations.	⊖ Might necessitate hard mental operations to understand the logic of the superimposed fragments (conditions, negations, loops).
Hidden dependencies	⊖ Most of the properties of the graphic components are hidden in the properties view. ⊖ Some information might be hidden because of the restrained area of the graphic components: use cases name, types of the attributes, parameters of the methods.		⊖ Might be difficult if the diagram contains a lot of interactions: the names of the source and destination lifelines might be hidden because they cannot fit in the same screen.
Premature commitment	⊕ Designers have the possibility to create parts of the diagrams in a random order. ⊖ The necessity of laying out the UML elements afterward to avoid the possible crossing edges might create a cognitive load.		⊖ Designers should care about the order of the events in advance.
Progressive evaluation	⊕ Designers can review the requirements with the technical team or with the client even if the use case diagram is unfinished.	⊕ Designers can see the evolution of their diagrams progressively: designers can generate code at any step of the diagram's creation.	⊕ The sequence diagram might be reviewed by the team members before its completion.
Role expressiveness	⊕ Stick-man is expressive.	⊖ UML interfaces, classes, enumerations, dataTypes are graphically similar. They can't rapidly reflect their roles.	⊕ Stick-man is expressive.

	Use case diagram	Class diagram	Sequence diagram
Secondary notation	⊕ Shapes, figures (icons) are the only visual variables used in the primary notations. ⊖ Color, size, orientation, brightness and grain are not used.	⊕ Shapes, figures (icons) are the only visual variables used in the primary notations. ⊖ Color, size, orientation, brightness and grain are not used	⊕ Position, shapes, figures are the only visual variables used in the primary notation. ⊖ Size, brightness, color, grain and orientation are not used.
Viscosity	⊖ The necessity of laying out the diagrams after a deletion/add of an object. ⊖ The modification of an attribute, method, properties of an edge should be performed in the properties view.		
Visibility	⊖ Visualizing the semantically linked elements with the other views is not directly visible. They are put in the properties view.		

As illustrated in Table 3.1, the CDs framework allows us to point out the possible sources of cognitive load in our chosen modeling environment: Papyrus and the three UML diagrams. Having a determinate number of dimensions to check our environment's effectiveness allows us to do a rapid evaluation. We are sure that we target the right dimensions and that an observed problem will almost certainly induce a cognitive load to our users. There is no doubt then that the CDs framework has indeed structured a great discussion support about programming environments (Around 1300 citations in Google scholar statistics since 1996). It might be considered as a starting point for later researches about programming environments despite of some limitations that we discuss in next paragraph.

Limitations and positioning

The CDs is only an evaluation framework. Our analysis of the three UML diagrams in the Papyrus environment identifies only the sources of the cognitive load in each dimension. The CDs framework does not provide us with guidelines to lighten such a revealed cognitive work. For example, the CDs does not mention how to put all the hidden dependencies as visible in the UML class diagram without overloading the human mind. In addition, it does not provide solutions to reduce the mental operations that are needed to understand the nested fragments in a sequence diagram. Moreover, mechanisms to enhance the visibility of the semantically linked elements from different diagrams are not discussed. Then, the CDs does not describe effective techniques to make the UML class diagram's elements (e.g., interfaces, enumerations) more role expressive. Finally it does not show how to effectively make the UML lifelines closer to the objects of the reality they represent. In that context, many options of solutions are available in the literature. For example, effective interaction techniques might be employed in order to decrease the cognitive load in the visibility dimension by providing facilities of navigation between different semantically linked diagrams. We may also take advantage of the human visual perception system, as it is the case of this thesis. Many other potential solutions might emerge in the future for that purpose (e.g., artificial intelligence).

The CDs framework mentions examples of solutions. Some of them take advantage of the visual system like for the secondary notation dimension where T. Green et al.

suggest to benefit from the additional graphic means to the formal syntax (e.g., color and layout). But, many fundamental details are not specified like recommendations for choosing the most visually selective colors or the difference between color and brightness (i.e., dark and bright colors). In addition, the effective implementations of color to UML nodes (i.e., to the border, to the text or both of them) are not discussed. Some implementations can create an additional cognitive work to the human readers instead of reducing it. Moreover, alternatives of color to take into account the possible color blindness of practitioners are not mentioned. In the visibility dimension, the authors propose the juxtaposition of parts of code or diagrams. However, if programmers juxtapose two complex and large diagrams, going back and forth each diagram might increase the cognitive load.

In this thesis, we explore the power of the visual system to reduce the cognitive load in some dimensions of the CDs framework. Particularly, we study the effective ways to employ all the available graphical means: the visual variables. Besides, the CDs authors have described the visual system as a powerful extra-channel. They regret its undervaluation: *Regrettably, this extra channel of communication between programmer and reader seems to have been little investigated.* Below are listed the CDs that our work tries to cover:

- Abstraction gradient: By grouping graphic components that share a particular property together, readers become able to visually control them as one entity.
- Closeness of mapping: Mapping information of the reality to the closest visual variable which will represent it reinforces the closeness of mapping in UML. For example, ordered information like the progress of the implementation of a project in a class diagram should be expressed by a visual variable which renders such order (e.g., a brightness variation from light to dark).
- Diffuseness: The graphical layers mechanism allows to make invisible some parts of the UML diagrams. In that way, the size of the diagram might be decreased to fit in a same screen.
- Role expressiveness: A role is attributed to each graphic component (or a group of them) by expressing a particular information in an UML diagram (e.g., a group of classes which are developed by a particular developer of a technical team).
- Secondary notation: Is the core dimension of this thesis.
- Visibility and juxtaposition: Semantic links between elements belonging to different diagrams of an UML model might be visually accentuated by the use of the same category of a visual variable.

The investigation of the power of the visual system has been also the core of the work of Daniel Moody as denoted next.

3.1.2 Physics of notations framework

The physics of notations (PoN) is a scientific framework to create new cognitively effective visual notations and to evaluate existing ones. It comes to explore the power of the visual system to lighten the cognitive load in some dimensions of the CDs framework. Moody has structured a scientific basis for visual notations in software engineering. In fact, the PoN is based on a wide range of scientific theories and empirical

researches. We will discuss the limitations of such high level guidelines in regards to their easiness of operationalization in practice [42].

The 9 principles of the PoNs framework

The Physics of Notations (PoN) framework [34] has been published by Daniel L. Moody in 2009. Moody has stressed on the importance of visual notations in software engineering field. He has pointed out that most of the existing works (at that time and till now) had been mainly spent in semantics of visual notations (i.e., the abstract syntax). Their visual syntax (i.e., the concrete syntax) had been little discussed and undervalued in their design level. The PoN comes to cover this gap. The name of this framework comes from the fact that its main focus concerns physical (i.e., perceptual) properties of notations rather than their logical (i.e., semantic) properties. Following are explained the nine principles of the PoN:

Semiotic Clarity is the first principle of Moody's framework. It denotes the necessity of having a one-to-one relationship between the semantic concepts of a visual language and the graphic concepts.

Perceptual discriminability denotes that every symbol of the visual syntax should be clearly distinguishable one from each other. Moody cites works from psychology and cartography to define what is called the visual distance. This distance might be measured by the number of visual variables used in the visual syntax and the number of steps of each one (e.g., red, blue and green are three steps of the color visual variable). The higher the visual distance is, the better the graphic objects of a visual notation are perceptually distinguished. Moody introduces 4 notions to decrease the visual distance between symbols:

- *Primacy of shapes*: Based on works around objects recognition, shapes are the primary basis to distinguish between symbols. Moody recommends the use of shapes from different families to increase the visual distance between them.
- *Redundant coding* refers to the use of multiple visual variables to distinguish between symbols (e.g., by using colors to distinguish between classes and edges in UML).
- *Perceptual pop-out* denotes the use of a unique value on at least one visual variable. Such configuration allows the symbol to be relatively rapidly detected by the human visual system, without conscious effort.
- *Textual differentiation* Moody recommends avoiding text to differentiate between symbols. He encourages using the visual variables instead. Using text is cognitively ineffective because it is serially treated by the human mind (i.e., contrarily to the visual variables treatment which is automatic and parallel).

Semantic Transparency denotes the use of symbols that reflect their meanings. If the *Perceptual discriminability* concerns the differences between the symbols, this principle relates to the relationship between the symbol and its semantic meaning. The higher the transparency is, the less the cognitive load to read or learn the representation is. The use of icons for instance helps speeding up their recognition. They are not

often used in software engineering. This principle mainly refers to the *role expressiveness* dimension of the CDs framework.

Complexity Management is related to the capacity of a visual notation to represent information without overloading the human mind. Moody emphasizes the existence of perceptual and cognitive limits of the human mind that visual notations designers should take into account. Diagram size and the number of diagrams elements affect respectively the perceptual and cognitive limits. Visual notations should provide mechanisms to manage the complexity in their primary notation. Moody proposes two ways to reduce the complexity of large systems: modularization and abstraction. Modularization means providing mechanisms in the primary notation that allow dividing large systems to small parts (e.g., packages in UML). Abstraction refers to the ability of a visual notation to represent a system in different levels of abstractions. This principle mainly refers to the *abstraction gradient* dimension of the CDs framework.

Cognitive Integration concerns only the systems which are represented by multiple diagrams. It denotes the ability of a notation to cognitively integrate information from the different diagrams. Moody differentiates two kinds of cognitive integration. The first one is at a conceptual level. It refers to the ability of a visual language to help building a coherent mental representation of the whole system from its different diagrams. The second kind of cognitive integration is the perceptual one. It denotes the perceptual cues to help in navigating between the different diagrams. This principle mainly refers to the *Visibility and juxtaposition* dimension of the CDs framework.

Visual expressiveness refers to the use of the full range and capacities of visual variables. Moody presents 8 visual variables: X and Y planar dimensions, size, brightness, color, texture, orientation and shape. He assimilates each visual variable to a communication channel that designers must fully consider. For example, the UML visual syntax mainly uses the shape visual variable (e.g., ellipses for use cases, rectangles for classes). It does not fully benefit from the other visual variables. Moody briefly describes the perceptive properties of these visual variables based on the SoG. This principle mainly refers to the *secondary notation* dimension of the CDs framework.

Dual Coding relies to the use of text to complement and reinforce the meaning of a graphic symbol. Combining both text and graphic symbols to convey meaning is better than using either one or another. Text and graphic symbols are differently treated by the human mind. Such combination strengthens the referential connections between them. Moody proposes to use annotations as well as comments to improve the understanding of visual notations.

Graphic Economy concerns the number of the graphical symbols used in a notation. This number should be cognitively manageable by humans. There is a cognitive limit beyond which the cognitive effectiveness is reduced. This limit is around 6. Moody proposes three methods to manage the graphic complexity of a notation. Reducing or partitioning the semantic complexity of a notation is the first strategy. The second strategy consists of not visually encoding all the constructs of a notation. Some constructs might be better expressed via only text. The third method is by taking advantages from the *visual expressiveness* principle. Combining the visual variables increases the perceptual discriminability and allows more manageable graphical symbols.

This principle mainly refers to the *Consistency* dimension of the CDs framework.

Cognitive Fit refers to the ability of a notation to fit to the different contexts of its use. Notations should provide different dialects to satisfy all types of audiences: Novices and experts. They should also manage the different possible contexts of use: communication, analysis, automation, etc.

Advantages of the PoNs framework

All the PoNs help visual notation designers to effectively translate their semantics into graphics. It yields also to have effective perceptions of the semantics of a diagram's elements. In fact, the PoN has appeared almost 20 years after the first publication of the CD framework. D. L. Moody denotes that the latter framework is not specifically focused on visual notations. He also mentioned that the CDs are not design guidelines and issues of effectiveness are out of the CDs framework's scope. Moody suggests to treat such issues via the PoN. For example, he defines the *visual expressiveness* principle to show how an effective use of *secondary notation* CD might decrease the cognitive load of humans.

Limitations and positioning

The principles are considered as high level guidelines. Störrle et al. [42] tried to use the PoNs framework to analyze the UML Use cases diagram. They chose only two principles: the *semiotic clarity* and *the perceptual descriminability* (i.e., because of space constraints). They conclude that the principles are ambiguous and assumptions for each principle are sometimes error prone. They also find that the principles are given without quantitative data. They assume that *the PoN, in its current form, is neither precise nor comprehensive enough to be applied in an objective way to analyze practical visual software engineering notations.*

Ambiguities in some of the PoN principles are due to the fact that scientific theories have been too summarized. We will illustrate such observation in two principles. In the *visual expressiveness* principle, the author had briefly mentioned the capabilities of each visual variable. Some ambiguities might occur here. For instance, the author does not precise if the number of perceptible steps of each visual variables apply equally to UML nodes and UML edges. In addition, he does not discuss the effective implementations of the visual variables to the graphic elements in software engineering. For example, the implementation of the size visual variable to an UML node might concern its area and/or its contained text (See figures in Chapter 4, Section 4.4). Concerning the *perceptual discriminability* principle, the author explains the primacy of shapes based on theories of objects recognition. Shape is the primary basis on which objects are identified. The author proposes then to use shapes from different families to distinguish graphic symbols of visual notations. But, the most effective families of shapes that can be used are not discussed. In the same principle, he explains that visual distance can be increased by combining visual variables in the same symbol. In this case, we wonder about the best combinations of visual variables that should be used and about the worst ones that might present risks of effectiveness and should be avoided. Finally, the impacts of such combinations on complex symbols like UML nodes (i.e., containing text) are not specified.

These ambiguities show that there are still multiple barriers which hinder the sys-

tematic definition of effective notations. Notably, in the following five principles that we target in our research:

- Perceptual discriminability: We provide guidelines of effectiveness which help increasing the visual distance between the graphic components in which a particular visual variable (or a combination of visual variables) is applied.
- Complexity management: The layers mechanism allows managing the complexity of big UML diagrams, by making visible or not some parts of the UML diagrams.
- Cognitive integration: Visually emphasizing the semantically linked UML elements of different diagrams helps enhancing the cognitive integration.
- Visual expressiveness: We define guidelines to effectively take advantage of the full range of the visual variables in UML.
- Cognitive fit: Using the visual variables to show the important elements in a communication, omitting information by putting them invisible via the layers mechanism are methods to make UML better fit with people who are not familiar with UML (e.g., clients).

We chose to investigate theories from the cartography field as defined in [10] to see if we can find more precise guidelines or quantitative metrics.

3.1.3 Summary

Table 3.2 below summarizes the CDs and the PoNs that our work targets. They are marked by Check-marks (✓). CDs and PoNs which are related to each other are put in the same lines.

Table 3.2: Positioning of the work of this thesis in relation to the CDs and the PoNs frameworks.

CDs framework		***PoNs framework***	
CDs	***This thesis***	***PoNs***	***This thesis***
Abstraction gradient	✓	Complexity management	✓
Role expressiveness	✓	Semantic transparency	
Secondary notation	✓	Visual expressiveness	✓
Visibility and juxtaposition	✓	Cognitive integration	✓
Consistency		Graphic economy	
Progressive evaluation			
Closeness of mapping	✓		
Premature commitment			
Diffuseness	✓		
Error proneness			
Hard mental operations			
Hidden dependencies			
		Dual coding	
		Cognitive fit	✓
		Perceptual discriminability	✓
		Semiotic clarity	

We have seen that for a designer of a visual notation who cares about the cognitive effectiveness of his modeling environment might use the CDs to check if the delicate points (i.e., cognitive load risks) are well treated. But, the "well treated" remains subjective because the CDs point out the risky areas without providing guidelines to resolve a revealed problem. Moody has proposed the PoN to remedy such weakness by focusing on visual notations and excluding textual ones and interaction mechanisms. His principles are very useful but they are sometimes incomplete and lead to ambiguities. That is why we will study a framework, the SoG, which is much more general while being much more precise and focused on UML.

3.2 The Semiology of Graphics (SoG)

Until the mid-60s, maps were generally only basic inventories and served as the artificial memory, whatever the disciplines where they were used[3]. During the 60s-70s, the quantitative geography and its complex maps appeared, maybe thanks to the relative availability of (very big) computers. It begun to become really important to answer to the following questions: How to draw a map? What do we have to print in order to effectively communicate, without loss of information? It is no coincidence that the Semiology of Graphics (SoG) has been published for the first time in 1967. The SoG is considered as one of the main theoretical foundations of the cartography science (and Information Visualization). It aims at helping cartographers or statistical analysts to construct effective graphical representations. The effectiveness of a graphical representation is defined as its capacity to provide the correct and complete answer to a given question. The answer has to be given in a relatively short time compared to another less effective graphical representation. The SoG is considered as the reference in the cartography field. In [29], Kosslyn reviews five books on graphics and charts (i.e., including the SoG) from the cognitive psychology viewpoint. The review was based on fourteen criteria. Most of them have been fixed based on the properties of the human visual processing system. For instance, he tries to assess if a book takes into account the short term memory constraints in the proposed theories. Three criteria concern the originality, the readability and the generality of each book. The results of the review show that the SoG is placed among the three first best books in terms of originality and generality [4]. It is also ranked first in all the cognitive psychology criteria. However, it is ranked last in the readability criterion [29]. The SoG is difficult to read due to the complicated taxonomies, philosophical reasoning and the crossing references of figures. The SoG is considered as *"unique"* because the author does not make references to any previous works. The only author he cites is Zipf to define the notion of effectiveness of visual representations. In fact, the theories described in the SoG represent the results of the author's long experience in cartography. It mainly reflects the results of his observations and manipulations of a large amount of graphic documents. In that context, some researchers reproach the lack of empirical validation of the SoG theories. As an attempt to validate these latter, Garlandini et al. [22] conducted a controlled experiment. They have proven that the theories of the SoG related to the visual variables and their characteristics are empirically validated.

[3]For example, the International Cartographic Association (ICA), the main scholarly organization related to cartography, is born in 1959.

[4]A simple search in google scholar, done by Gill Palsky in 2017, a professor of cartography, in a special tribute to Jaques Bertin in the fifty years old of the SoG. It showed that the SoG is still be the most cited book among six other books of cartography

The SoG exhaustively treats all the possible types of graphical representations: maps, networks and diagrams. It provides objective rules to support, using graphics, each step of graphical representation design. It covers the data analysis process, the visual representation of data and finally the reading process. That aims at providing a set of mandatory rules to control the construction of effective graphical representations.

In this section, we will expose the basic theories of the SoG and, in parallel, position UML in relation to each notion. This section represents a first refinement layer to adapt the SoG principles to UML.

3.2.1 Graphical representations: Maps, diagrams and networks

The scope of the SoG is delimited to graphical representations which might be transcribed: in a white plane page, having a medium size, in a normal lighting and using the available graphic means. The kind of a graphical representation depends on the nature of the information to transcribe in the two dimensional plan. Let us assume that a designer wants to represent the registration frequency in his e-learning platform. For that, he queried the corresponding database and got the information represented in Table 3.3.

Table 3.3: Example of data for visual encoding.

Date	Day 1	Day 2	Day 3	Day 4	Day 5
Number of registrations	75	50	10	25	5

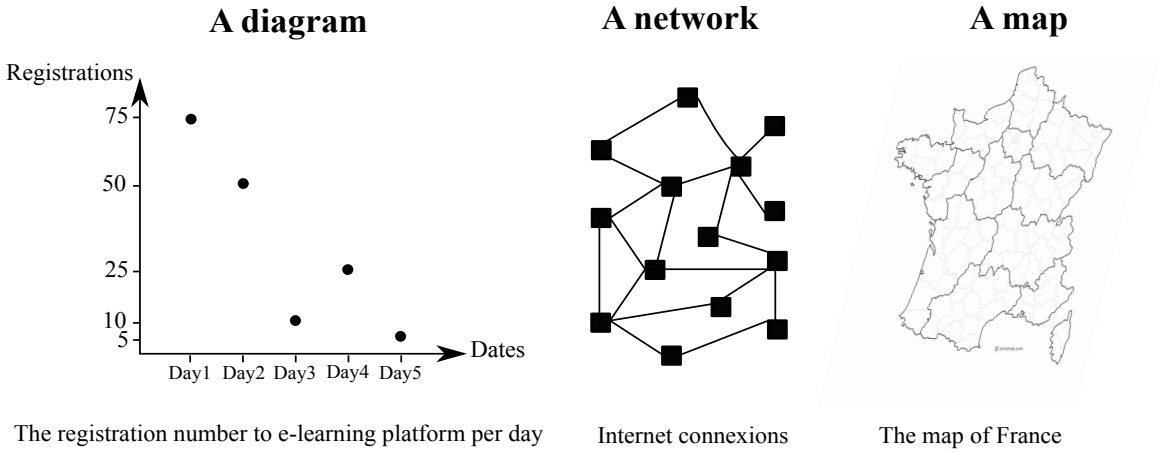

Figure 3.1: The three types of graphical representations

For each date, there is a corresponding number of registrations. The adequate visual representation is then a diagram, as shown in Figure 3.1. A diagram is the adequate visual representation when it comes to transcribe two different information properties (here the dates and the registration numbers) which are related to each other. Now, if an information describes relationships between elements of the same nature, the adequate representation is a network. An example of network is illustrated in Figure 3.1. It might represent the courses which are frequently chosen together by the students. Finally, if a designer wants to transcribe geographic data, a map is the adequate graphical representation (Figure 3.1). Here, it might transcribe the registration frequency per city.

Positioning of the graphical representations in UML: In software engineering, most of the graphical representations are considered as networks. They are made of graphic elements which have the same nature and which are linked to each other using edges. In UML, we find classes which are related to each other using edges to form a class diagram, use cases which are linked to each other to constitute a use case diagram, activities which are connected to each other to compose an activity diagram, etc. UML diagrams might also be seen as maps, as shown in Figure 3.2, where just like cities, UML graphic components have a particular position (e.g., defined by an automatic layout algorithm, imposed by the logical and chronological order of the events) and a particular size (e.g., might depend on the contained text length).

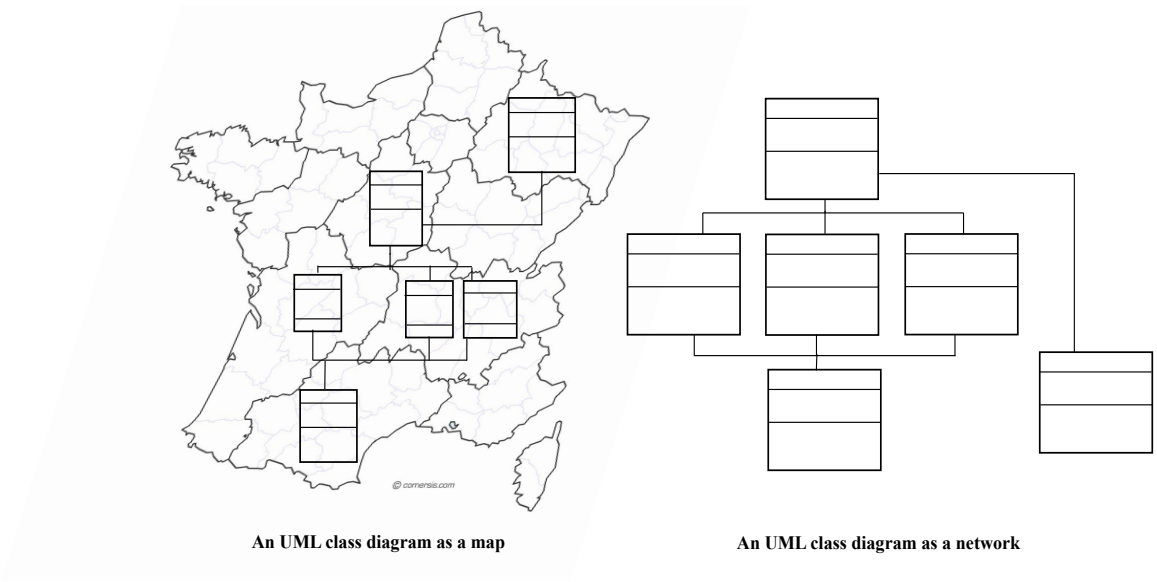

Figure 3.2: An UML class diagram: A network or a map

3.2.2 Points, lines and zones

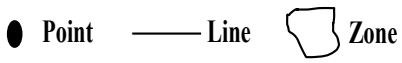

Figure 3.3: Points, lines and zones of the SoG.

In the three types of graphical representations, different spots can be employed. The SoG pragmatically identifies the forms in which any spot might be in correspondence to the two planar dimensions (X and Y). In fact, as illustrated in Figure 3.3, a spot might be a line, a point or a zone. These three forms represent the elementary spots of the geometry. If a spot has a significant length but with no area, it is a line, the equivalent of a line segment in geometry. If a spot has a non significant length and a non significant area, it is a point. Points are mainly characterized by their position in correspondence to the X and Y planar dimensions. Whatever the spot which represents it is, only the center of this latter is significant. Finally, if a spot has a significant (fixed) and measurable area, it is a zone. A city in a map is considered as a zone: it has a significant area that is proportional to the real surface of the city and has a fixed position that should accurately reflect its position in the reality. That is why map drawers cannot change the position and the area of a zone.

Positioning of the graphic spots in UML: In UML, it is clear that edges correspond to lines of the SoG. The other graphic components are considered as points: they have non significant areas. In fact, difference of the area between the graphic components in an UML diagram is not significant to the reader. For instance, such difference might exist due to the different lengths of the contained text. Classes are then considered as points, as well as use cases, classes, packages, etc,. Sometimes the position of these latter is fixed to respect a meaningful order (e.g., states in a state machines diagram). UML graphic components are then considered as points which might sometimes have a fixed position.

3.2.3 Organization levels of information properties

Information is the key element of knowledge that a graphical representation transcribes. For example, the diagram in Figure 3.1 above represents the following key information: *The registration numbers to the e-learning platform per day*, the map in the same Figure illustrates *the registration frequency to the platform per city* as a key information. Each information contains one or more than one concept which can vary. For example, in the first information, we find two concepts of variation: *the registration numbers* and *the dates by days*. The registration numbers change depending on the dates which also vary, by a unity of a day. These concepts are called information properties (i.e., they characterize a particular information). In that context, graphical representations serve to encode information via its properties in a two dimensional plan. The categories of an information property (e.g., days 1 until day 5 are the categories of the information property date) can be organized differently. They can be ordered, qualitative or quantitative which refer to what are called organization levels. The organization level of an information property refers to the way in which the corresponding categories are organized between each other. Like in statistics, an information property might be respectively ordinal, nominal or interval. The three levels are explained below one by one. They will be illustrated based on the information properties that practitioners needed to visualize in their practice of UML (See Chapter 2).

Qualitative organization level

Information properties are considered as qualitative if their categories are different from each other and can be ordered in different manners without inducing to ambiguities. In addition, they are equidistant.

Parts of a pattern (e.g., MVC), parts of layers, ownership of each class are three qualitative information properties. Their categories are equidistant and might be ordered in different manners. *Eva, Steven and Deve* are three categories of the qualitative information property *ownership of classes*. They are equidistant and can be re-ordered based on the alphabetic order: *Deve*, *Eva* and then *Steven*.

Ordered organization level

An information property is considered as ordered if its categories are ordered in one single and universally recognized manner. They can be compared to each other and allow to state that a particular category is more or less/before or after another category.

Criticality, levels of security, progress in testing, in implementation or in execution are ordered information properties. Their categories are organized in an universal order. For instance, *Implemented, in progress and to implement* are three ordered categories of the information property *progress in implementation*. Their re-organization is a source of ambiguity.

Quantitative organization level

The quantitative level concerns countable categories which can be compared to each other. It allows to state that a particular category is the double of that one, the half, four times that, etc. Broadly speaking, it reflects the ratio between its categories.

The information property *Quality metrics* is quantitative. An example of a metric might be the total number of methods of an UML class [43]. The related categories are positive integers which might be compared to each other. It might be insightful to see that a class has, for example, three times the number of methods compared to another class.

Table 3.4 summarizes the information properties which has been discussed according to their organization levels. Each information property can be characterized by only one of the three possible organization levels defined by the SoG.

Table 3.4: Organization levels of information properties in UML.

Organization levels/ Information properties	*Qualitative*	*Ordered*	*Quantitative*
Criticality		✓	
Parts of a pattern	✓		
Parts of layers	✓		
Levels of security		✓	
Progress in testing, implementation, or execution.		✓	
Ownership of each class	✓		
Quality metrics			✓

3.2.4 The visual variables

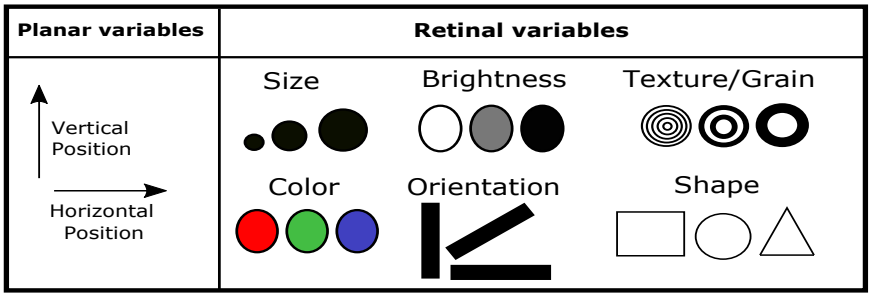

Figure 3.4: Visual variables

For each spot (a point, a line or a zone), authors of a graphical representation can

define its position according to the two planar dimensions (X and Y). Also, they can change its visual aspect by changing the size, brightness, grain, color, orientation or shape. All of these visual characteristics of a spot are called the *visual variables*, in the SoG jargon (Figure 3.4). The visual variables except the two planar dimensions are called *retinal variables*.
Following are the definitions of each retinal variable (Figure 3.4 can help to better understand them):

- **Size**: **The variation of the area of a spot represents the visual stimuli to the size variation.**
  Any spot, having a punctual or a linear signification, can vary its size without changing its position, brightness, grain, color, orientation or shape.

- **Brightness**: **Continuous progression that the eye perceives in the series of grays which ranges from black to white.**
  This progression is independent from the color and we can range from black to white by different levels of gray, blue, red, etc. Only one color should be used for such transitions.

- **Grain**: **The grain is the succession of photo-graphical reductions of the similar spots that are contained in a particular texture.**
  In a specific surface and for a particular texture, these reductions increase the number of spots, without varying the brightness. The grain is the quantity of separable spots contained in a unit surface. Broadly speaking, we can say that the grain corresponds to the magnification factor applied on the texture (if any). In a particular brightness, the grain is the quantity of separable spots contained in a unit area.

- **Color**: **The color variation is the differentiation generally induced by different colored excitations perceived between two ranges having the same brightness.**
  Human readers cannot understand the colors sensations if, first of all, the notion of color and the notion of brightness are not rigorously differentiated. Color and brightness are two different retinal variables. Each one is characterized by its perceptive properties.

- **Orientation**: **A spot, in the form of a point, line or zone, can take an infinity of different orientations without changing its center.**
  We can feel the differences of orientations as far as the spot represents a linear aspect (i.e., the ratio Height/basis have to correspond to at least 4/1). The difference of angles between multiple parallel spots constitutes the stimuli of the orientation variation.

- **Shape**: **A spot having a constant area can take an infinity of shapes (e.g., rectangles, circles, triangles).**

To better take advantages of the retinal variables and ensure the effectiveness of graphical representations in which they are used, it is important to study their characteristics and the rules guiding their use. All UML users and tool vendors should take them into account while respectively providing and using these retinal variables.

Visual variables and UML

For UML graphic nodes, the shape is the mainly used retinal variable. It includes rectangles, ellipses, squares and circles. The brightness applied to the border is used to distinguish between inherited states in state machine diagrams. The grain retinal variable is also used in the borders of some UML graphic nodes via dashed and continuous border lines. In fact, as illustrated in Figure 3.5, a continuous line might be seen as the result of a zoom on a dashed line.

Concerning graphic edges, we find that the grain, the shape and the brightness are the

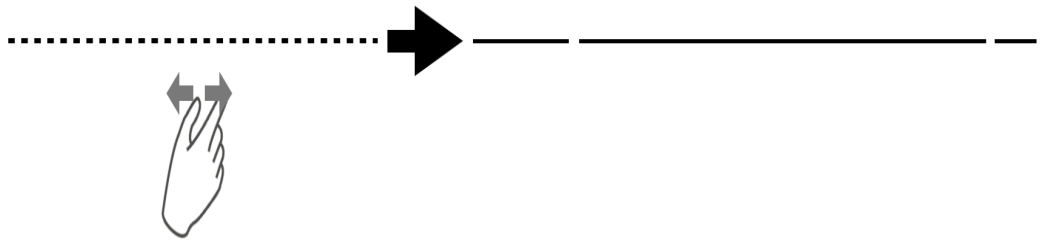

Figure 3.5: Continuous and dashed lines: a grain variation

only used retinal variables. First, UML graphic edges might be expressed via dashed lines or continuous ones, which corresponds to the grain variation. Second, the shape variation is used at edges extremities, in the center or above them. We find triangles, circles, diamonds and squares which might be empty or filled. This corresponds to a brightness variation.

The retinal variables are not fully explored by the UML concrete syntax (e.g., orientation, size, color). That leaves the non used retinal variables under modeler's control. Size, brightness, grain, color and orientation might then be used, as long as they do not alter the UML primary notation. For example, the grain of UML nodes borders cannot be changed because it has a semantic significance in UML (i.e., for inherited states). However the grain of their background can be changed without altering the UML visual syntax. The control is not totally explored by UML practitioners. In fact, the retinal variables are already little used. We observe that color is the mainly used one (Chapter 2).

3.2.5 Perceptive attitudes of the visual variables

If the information properties might be differently organized, it is likewise for the visual variables. Their organization level refers to the perceptive attitude taken by the reader when she/he perceives its variation. In fact, each human reader will spontaneously order a series of values of brightness starting from the white to the black or from black to white but never in an other different order. However she/he can order a series of shapes in different manners, no visual order is fixed. The SoG distinguishes four perceptive attitudes: associative, selective, ordered and quantitative. Below, we explain each attitude:

Selective perceptive attitude

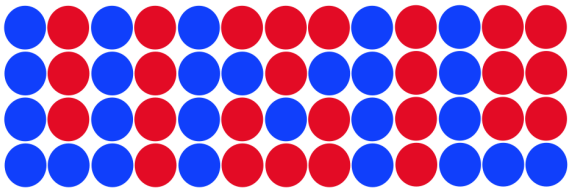

Figure 3.6: Selective perception with colors

In the selective perception, the human reader can visually select/isolate all the spots belonging to the same visual category (e.g., same color, same size) and visually abstract all the other ones. She/he can then perceive only the requested group. By decorating spots with a selective visual variable, they will form different groups relatively easily identifiable and the reader can visually select the group that she/he wants to see. In Figure 3.6, the human eye (you) can easily visually select the group of blue points from the red ones and read the word "UML". Selective attitude allows the readers to answer questions about one single spot in a graphical representation like *Where is the lifeline named X?*. It also helps the readers answering questions about a group of spots like *Where are all the MVC*[5]*-compliant controllers classes*? This last perception can be instantaneous if the used visual variable is selective. Otherwise, the perception can take a relatively long time because the reader will search spot by spot until finding her/his response.
Color is recognized to be a selective retinal variable, like the size, the brightness and the grain. The orientation is selective only for lines and point. Zones having different orientations are not selective.

Associative perceptive attitude

The human reader can spontaneously associate all the similar spots into one single group, despite of a particular visual variable variation. In Figure 3.7, the reader's eye can associate all the spots in one single group thanks to the associative characteristic of the shape. They are perceived as similar.
Association is required if the graphical representation combines two information prop-

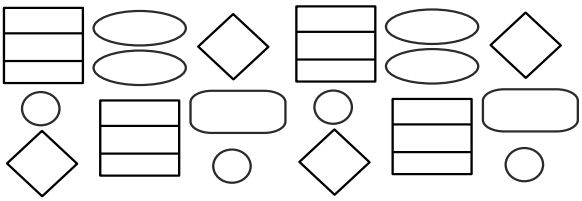

Figure 3.7: Associative perception of shape

erties. The reader's eye will relatively easily associate each representation of information property with a particular associative visual variable, all categories combined (e.g., all colors combined, all shapes combined). For example, the human eye can visually associate all the use cases together despite of a color variation on them: color is an associative retinal variable as well as the shape and the orientation.

[5]Model-View-Controller design pattern

A dissociative visual variable, contrarily to an associative one, forbids the visualization of all its categories combined. But also, it annihilates the perceptive attitudes of the other visual variables when combined with it. For example, it might forbid to operate a spontaneous visual selection of colors when they are combined with it. In fact, it dominates all the combinations of visual variables done with it. This represents what is called dissociation.

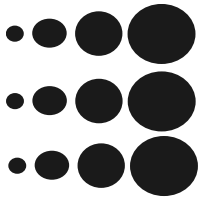

Figure 3.8: Dissociative perception of the size

Figure 3.8 illustrates a series of points with different categories of size. The reader's eye cannot associate all the spots in one single image. Contrarily to associative visual variables, the different points do not have the same visual power, the same visibility. This is due to the the difference of quantities of black in each spot.

Size and brightness are both dissociative retinal variables.

Ordered perceptive attitude

Ordered perception is used when the reader wants to compare two or more than two categories. Such comparison is spontaneous, if the used visual variable is ordered. Otherwise, it will require a careful analysis and the used visual variable is considered as non ordered.

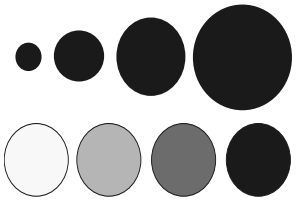

Figure 3.9: Ordered perception of the size and the brightness

Figure 3.10: Non ordered perception of colors.

It is clear, in Figure 3.10, that colors are not ordered. Each reader can establish a different order, no spontaneous order is imposed. However, the size and the brightness visual variables (figure 3.9) force immediately a significant and a universal order (e.g., from the biggest spot to the smallest on or inversely). They allow the human reader to state that a spot is bigger or brighter than another.
The categories of an ordered visual variable might also be seen as only different. They ensure the selective perceptive attitude.
Three retinal variable are recognized to be ordered: size, brightness and grain.

Quantitative perceptive attitude

Quantitative perception is used when we search to define, using numbers, the ratio between two spots. When the perception is quantitative, the numerical ratio between two spots is immediate.
The reader can perceive that a line length equals three times another line length, that an area is 1/4 of another area, etc.

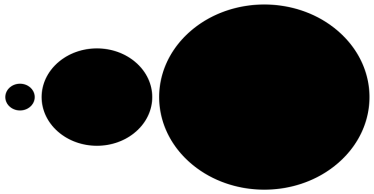

Figure 3.11: Quantitative perception of the size

Figure 3.11 shows a series of points with a size variation. They were created based on the abacus of sizes that has been created by Jaques Bertin. The reader can spontaneously perceive that the second point is two times the smallest point, the third point is two times the second point. The size retinal variable is the only quantitative retinal variable.

Table 3.5 resumes the organization levels of the visual variables. The two planar dimensions are the only visual variables which benefit from all the organization levels. Empty cells mean that the corresponding visual variable does not guarantee the corresponding organization level to answer questions about a group or all the spots of a graphical representation (i.e., spots that are concerned by a variation of the corresponding visual variable). They guarantee the corresponding organization level only when the reader looks for one single spot.

Table 3.5: Organization levels of the retinal variables.

	Associative	Selective	Ordered	Quantitative
Size	×	✓	✓	✓
Brightness	×	✓	✓	
Grain	✓	✓	✓	
Color	✓	✓		
Orientation	✓	✓[6]		
Shape	✓			

Positioning of the perceptive attitudes of the visual variables in UML

UML diagrams are mainly composed of shapes which are related to each other using edges. As described earlier, the shape is an only associative retinal variable (i.e., that means that any type of variation can be applied without constraints). Its use in UML allows the human reader to associate all the graphic components in one homogeneous image. However, it also does not allow the human reader to rapidly visually select the

[6] Except in zones

similar shapes together. Such visualization could help finding relevant information in a particular diagram.

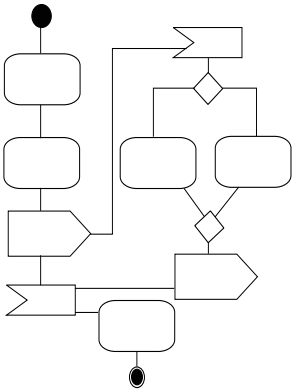

Figure 3.12: Empty activity diagram: A default grain variation

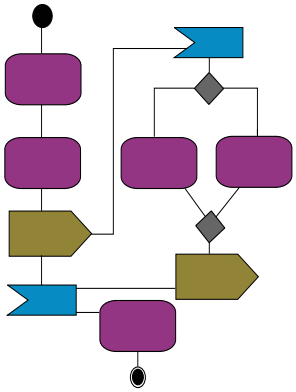

Figure 3.13: Selective perception of colors in an empty activity diagram

Figure 3.12 illustrates an empty skeleton of an activity diagram [7]. Practitioners might need to find a particular activity to accomplish their tasks. Visually grouping all the activities together (i.e., using a selective retinal variable) might help targeting such particular group and rapidly find the sought information. In Figure 3.12, there are 5 different shapes: circles, rectangles, diamonds and two other shapes (i.e., related to a trigger and a sendSignalAction). The human eye (you) cannot rapidly visually select the rectangles together, the diamonds together, etc. However, it can rapidly select the circles against all the other shapes. In fact, there is a grain variation in this diagram (Figure 3.12). Three categories of grain are employed: empty shapes, black circle and a black circle with a white inner circle (the end point). Such use allows the human eye to rapidly select the beginning and the end points of the diagram. But, it forbids the spontaneous selection of the similar shapes together. The shape and the grain are the only retinal variables used by the activity diagram's primary notation. Now, if a designer wants to allow the reader to visually rapidly select the similar shapes together, he can use the size, the brightness, the color or the orientation retinal variables (i.e., the grain and the shape are already used). In Figure 3.13, the color is used to group the similar shapes together. Designers should be careful about the use of the size and the brightness retinal variables which are both dissociative (i.e., they cancel the power/benefits of the other potential visual variations).

3.2.6 Capacity of the visual variables

The capacity of a visual variable defines the number of categories that a visual variable is able to effectively identify. Effectiveness here means that the reader's eye can perceive the used number of categories in a relatively short time compared to another bigger number of categories (e.g., the human eye can rapidly select a set of lines with 4 different sizes compared to another set of lines with 6 different sizes [10]).

The capacity of a visual variable depends on the organization level it is used for. In fact, in associative, ordered and quantitative levels, the capacity of a visual variable is unlimited from the pure graphical point of view but limited from the human eye

[7] At this level, we treat simple shapes as studied in the SoG. The containment of text in the UML graphic components is treated in the next chapter.

capabilities point of view. For example, the human eye cannot perceive more than 20 categories between two points having a ratio of 1/10. However, in the selective level, the capacity of each visual variable depends on whether the spot is a point, a line or a zone. Table 3.6 summarizes the capacities of all the retinal variables in the selective perception. Graphic authors should not exceed these numbers to ensure a rapid perception of the categories of each retinal variable.

Table 3.6: Capacity of the retinal variables in the selective perception.

	Size	**Brightness**	**Grain**	**Color**	**Orientation**
Points	4	3	2 to 3	7	4
Lines	4	4	3 to 4	7	2
Zones	5	5	4 to 5	7	-

3.2.7 Creating effective graphical representations: analysis and reading processes of the SoG

In this subsection, we explain the two fundamental theories of the SoG that help graphic authors creating effective graphical representations: the analysis process and the reading process.

Analysis process

All the previously described basics from the SoG help graphic authors building effective graphical representations. In fact, before any construction of a graphical representation, an analysis of the information to represent has to be performed, by identifying two important notions. The first one is the main information that a graphical representation is meant to represent. Thereafter, graphic authors should identify the information properties which might have different variations in the graphical representation.

Table 3.7: The population density per city in France in 2009

Cities	City 1	City 2	City 3	City 4	City 5	City 6	City 7	City 8	City 9	City 10	City 11	City 12	City 13	City 14	City 15
Inhabitant per km^2	0-5	5-15	15-30	30-50	50-80	80-110	110-150	150-250	250-500	500-1000	1000-2000	2000-4000	4000-8000	8000-15000	15000

Let us assume that we will graphically represent the data in Table 3.7. The main information consists of *the population density per city in France in 2009*. The properties of this latter are two folds: the cities and the population density in terms of number of inhabitants per Km^2.

For each information property, graphic authors have to identify its organization level and its corresponding categories. In our example, the cities are geographic data which should be transcribed in the map in figure 3.14. The population density has an ordered organization level and it holds 15 categories.

The organization level and the number of categories of an information property control the choice of the most effective visual variable. It is important to chose the visual

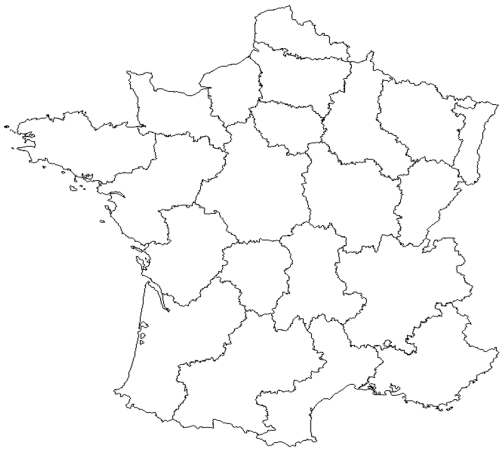

Figure 3.14: The map of France.

variable which has at least the same organization level as the information property it will represent. Obviously, all the visual variables do not suit to all types of information properties [10]. In fact, the human eye cannot perceive an order if the used visual variable is not ordered; it cannot perceive a ratio if the visual variable is not quantitative. The brightness is an ordered retinal variable. It has been employed to represent the ordered information property *number of inhabitants per* km^2, as illustrated in Figure 3.15.

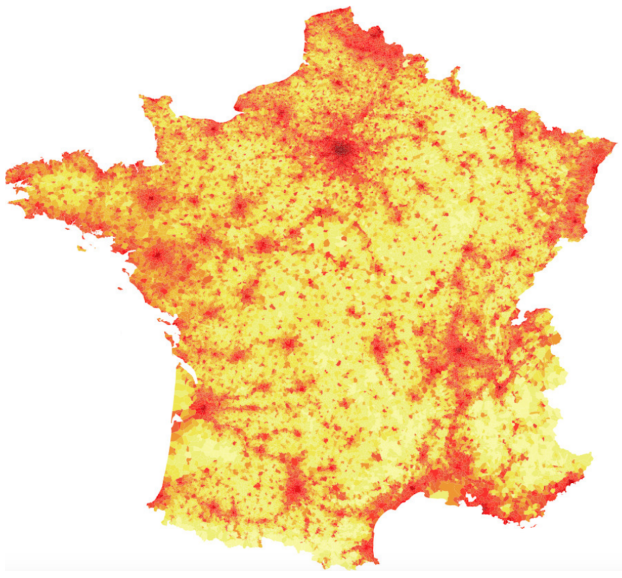

Figure 3.15: The brightness to express the population density.

But, such representation is not yet finished and the reader cannot understand the brightness variation. Apart from recognizing that the map corresponds to the map of France, the human reader cannot rapidly guess that the brightness variation represents the population density. He also cannot relate it to a particular period of time. He might question: *What does a dark orange means? What is the main information that this map communicates, the actual temperatures per city, the number of immigrants per city, etc? In which period these information have been collected?.*

That is why the analysis process is a primordial step that should not be undervalued. It helps producing two important outcomes: a title and keys (also called captions or

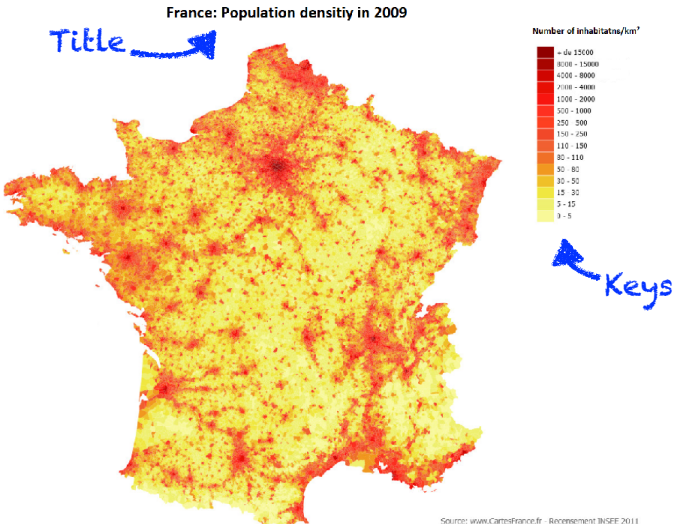

Figure 3.16: A map with a title and keys.

legends). They are primordial to ensure a relatively rapid reading process. The title should mention the main information that a graphical representation describes (i.e., the first element that has been identified in the analysis process). The keys indicate the information properties which are expressed in the graphical representation and their corresponding visual variables variations.

Reading process

The outcome of the previous analysis process are graphical representations, like the map in Figure 3.16. The SoG details the reading process of these graphical representations by the human readers. In fact, reading a graphical representation involves three successive steps called: the external identification, the internal identification and the perception of the information itself. If the external identification concerns the comprehension of the information that a graphical representation reflects via the title. The internal identification concerns the understanding of the mappings between information properties and their corresponding visual variables via the keys. The human reader become then able to perceive the information in a graphical representation to answer questions that he might need to answer to accomplish his task. Besides, based on the SoG, graphic authors are able to determine all the questions that a human reader might ask in front of a particular graphical representation. The title and the keys allow decreasing the ambiguities which might occur when reading a graphical representation. They allow decreasing the time necessary to read a graphical representation which is a major factor of effectiveness.

Analysis process and reading process in UML A strongly related work in this field does exist in the literature [23]. Genon et al. relate to the SoG to define guidelines in order to build effective keys in diagrams in software engineering. Their study of the existing literature revealed three main articles studying the effective keys in interactive environments. Interactive environment here means *a sub-category of dynamic environments where visual representations contain animations and the tool hosting the diagram is equipped with functions allowing the user to directly interact on the diagram.*

[17] describes guidelines to design effective keys for maps in dynamic environments, which is not the actual case in software engineering. [44] also defines guidelines for maps but makes a greater focus on user interactions. The work of Genon et al. is complementary to [39] which relates to the SoG to show that such theory can be used to empirically assess keys in dynamic environments.

Genon et al. refer to the SoG to help defining static keys for diagrams in software engineering. They deem that keys are crucial for a relatively immediate diagram understanding. Three scenarios where conceived to demonstrate the applicability of their guidelines in software engineering diagrams. The scenarios range from the impossibility of modifying the primary notation of the language to the possibility of completely changing it through a medium level of freedom of its modification.

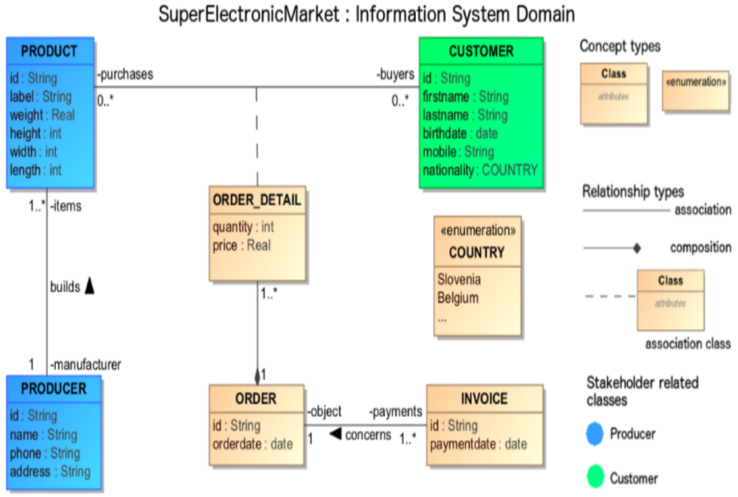

Figure 3.17: Effective keys by Genon et al. [23]

Figure 3.17 illustrates the result of the execution of their recommendations on an UML class diagram. A title and keys are embedded to the diagram. The title denotes the main information of the class diagram *SuperElectronicMarket: Information System domain*. The keys cover not only the secondary notation via the illustration of the meaning of each color variation (blue for producers and green for consumers) but also the UML primary notation where the semantic concepts are explained.
This work settles for prescribing guidelines to create effective keys in software engineering. It does not discuss the effective implementations of the visual variables on the software engineering graphic elements. For instance, the rational behind the application of colors to the background of classes is not explained. In addition, guidelines to help choosing the effective categories of the visual variables are not discussed. In Figure 3.17, the authors chose the blue and the green colors together to express the stakeholder related classes. We wonder then about the reasons of such choice and the degree of its effectiveness.

3.2.8 Effective categories of the retinal variables

This section answers the previously asked questions by providing methods which allow to choose the effective categories of each retinal variable. By effective categories, we refer to those that the human eye can relatively easily and rapidly differentiate the one from another. For example, the human eye can relatively rapidly differentiate between a set of black and white points than two gray points with different levels of brightness, as illustrated in Figure 3.18.

Figure 3.18: The notion of effective categories of the retinal variables: brightness.

Effective categories of the color retinal variable

To choose the effective categories of color, graphic authors should differentiate between the notion of color and the notion of brightness. They are two different retinal variables which are characterized by different perceptive properties. The following formula better illustrates such difference. For that reason, in a diagram, if colors are used and if they present the same brightness, the reader will benefit from the selectivity induced by them. But, if there are different brightness among them, another type of organization -coming from the brightness- will be perceived: the order. If the author of the diagram does not intend to bring some elements to the front and others to the back, she/he has to be careful about such use and the non wanted mix-up between color and brightness.

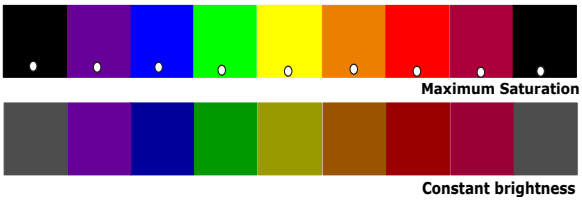

Figure 3.19: Saturated colors do not have the same value of brightness.

In that context, using colors from the spectrum (violet, blue, green, yellow, orange, red, purple and/or gray) might induce to an unwanted visual order. In fact, they have different brightness depending on the color, as illustrated in Figure 3.19 above. They are called saturated colors because they are pure ones. They contain neither black nor white (or a very small quantity).

As depicted in Figure 3.20 below, the right and the left regions of the spectrum, starting from the yellow, represent an ordered set of colors. The selectivity is in its maximum near to the saturated colors (colors of the spectrum and contain a white point in Figure 3.20) and decreases by moving away (Figure 3.20). The choice of the selective colors is different depending on the brightness.

- **In big brightness (i.e., light colors):** colors are chosen around the yellow, from green to orange. Light blue, violet, purple and red are less selective (Figure 3.20).

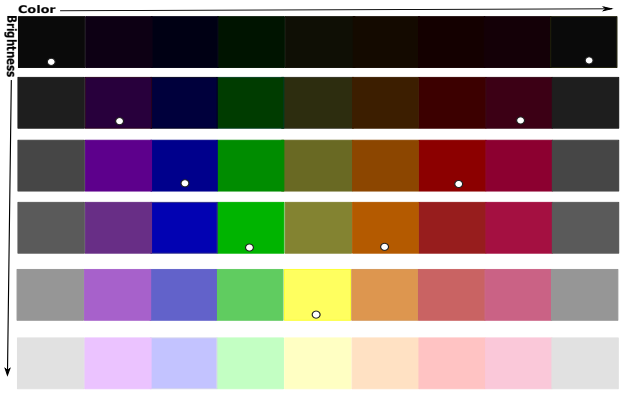

Figure 3.20: Colors of the spectrum in different brightness. Saturated colors are marked by a white circle.

- **In medium brightness:** we find the maximum number of selective colors. The blue and red colors are diametrically opposite (Figure 3.20): violet, blue, green, yellow, orange, red and purple might be used in this level of brightness.
- **In low brightness (i.e., dark colors):** the selective categories of colors range from blue to red by the violet and purple. Dark green, yellow and orange are less differentiated (Figure 3.20).

Following are described technical information about choosing the effective categories of colors:
If a designer wants to effectively choose his selective colors based on the previous method, he should use the Hue, Saturation, Brightness (HSB) system.
Light colors have a value of brightness (B) which is bigger than 70% (i.e., >= 150).
Medium levels of brightness are situated between 50% and 69% (i.e., 110<= B <150).
And, dark colors (i.e., low brightness) have a value of brightness which is lower than 50% (i.e., <110).

Color and daltonism Color is an excellent selective retinal variable. It can be easily combined with the other visual variables and it is rapidly memorable. It has an absolute psychological attraction. In fact, colors capture the attention of human readers. But, the use of color is not suitable for color-blind people. Indeed, anomalies of chromatic perception (i.e., color blindness) are more frequent than we usually believe. Color blind people compensate this anomaly by searching, usually unconsciously, alternatives of color to decrypt messages in a graphical representation. In that context, we mention that there exist three types of daltonism [7]. We find the *monochromatism* which consists of the total absence of the perception of colors. It is very rare (it affects A person per 40 0000). People having *monochromatism* perceive only black and white. The second type of daltonism is the *Dichromate*. It refers to the ability to perceive only two colors from the three primary colors (red, green and blue): either only green and blue or only red and blue or only red and green. The last type of daltonism is what is called the *anormal trichomate*. It consists of weaknesses in the perception of red, green or blue. Persons having this trouble cannot visually differentiate between these three colors. Graphical representations have to provide alternatives of colors to remedy for such possible problems (i.e., by making use of the other visual variables).

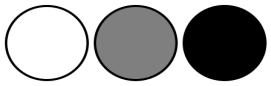

Figure 3.21: Three effective categories of brightness.

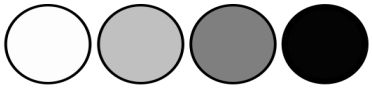

Figure 3.22: Four effective categories of brightness.

Figure 3.23: Effective categories of brightness.

Effective categories of the brightness retinal variable

To choose effective categories of brightness, the only constraint which has to be respected is to employ equidistant ones. We find different sets of effective categories which are recommended by the SoG as ready-to-use ones. As illustrated in Figure 3.21, for three categories of brightness: black, medium gray and white should be used. For four categories of brightness, black, two levels of equidistant gray and white, as shown in Figure 3.22.

More technically, in order to generate a formula for finding equidistant categories of brightness, Jaques Bertin found that the only condition is to attribute a value different from zero to white and a value different from one hundred to black (which corresponds obviously to reality). In that context, if we denote:
W: The percentage of the White in the spot, W $\neq$ 0.
B: The percentage of the Black in the spot, B $\neq$ 100.
r: The common distance between the chosen categories of brightness. It denotes the ratio between the values of brightness of two successive categories (i.e., a logarithmic value). $r \in \mathbb{N}$.
n: Number of categories, $n \in \mathbb{N}$.

It is clear that we have to multiply the value of white W by the common difference r the number of wanted categories n minus one times. Therefore, Bertin defines the formula below:

$B = Wr^{(n-1)}$.
As a result, the common difference r will be:

$r = \sqrt[n-1]{(B/W)}$.

Effective categories of the orientation retinal variable

The effective categories of orientations depend on the type of the spots. Below, these categories are explained:
Two categories in lines: the line (0 degree) and its perpendicular (90 degree), as illustrated in Figure 3.24.
Four categories in points: the horizontal (0 degree), the vertical orientation (90 degrees) and oblique orientations of thirty and sixty degrees, as shown in Figure 3.24.

Effective categories of the grain retinal variable

No particular method is described by the SoG to build effective categories of grain. The only recommendation is to choose equidistant categories where the percentage of

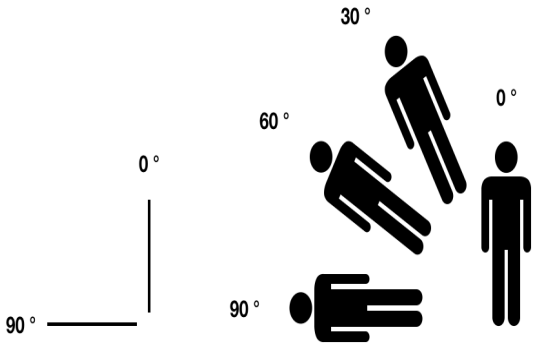

Figure 3.24: Effective categories of orientation for lines and points.

black is the same in all the spots. In a diagram, it is preferable to avoid using low values of grain (i.e., a lot of small spots), they will be altered after printings.

Effective categories of the size retinal variable

In a graphical representation, if the size retinal variable is used to express a quantitative information, the categories of this latter should be proportional to the expressed quantities. For that, the radius of a circle, the side of a square or a triangle should be varied. The SoG provides a well established abacus to choose the circles which correspond to a each quantity of an information property.

The effectiveness of the retinal variables categories in UML

In practice, most of the used colors are those of the spectrum. For example, according to the interviews (Chapter 2), practitioners make use of the blue and the yellow to respectively express tested and non tested functions. They also use the orange and red to respectively express the material related classes and those related to the system. In the models repository, the following ranges of colors were employed to express the MVC elements: (pink, yellow and mauve), (green, yellow and mauve), (blue, orange and green) and (yellow, green and red). Such usages are considered as non-effective from the SoG viewpoint. They are characterized by different brightness while the expressed information are only selective. In that context, we find that most of the modeling tools, if not all, do already not differentiate between color and brightness. They provide both of them to users via the same window, which might represent a reason of such non-effective usages. Concerning the other retinal variables, we find that they are not used in practice. Therefore, we cannot comment on the effectiveness of their categories. They are not yet handled by UML modeling tools.

3.2.9 The problems of applying the SoG to UML

In this section, we presented the main guidelines of the SoG that are related to the visual variables and their use to construct effective graphical representations. In that context, a correct analysis process of the information properties to be transcribed is a primordial step. The analysis process consists of mapping each information property to the most convenient visual variable. The mapping has to be done based on the perceptive properties of the visual variable and the organization level and the number of categories of the information property in question. Another basic step is the

annotation of the graphical representation with two elements: a title and keys. Both elements help fastening the reading process of the resulting representation.

On one hand, we can state that UML does not take advantages of the whole set of the visual variables in its primary notation. If the shape represents the most used visual variable, the grain and brightness are very less employed. The use of the other visual variables is usually left under modeler's control. On the other hand, points, lines and zones of the SoG are considered as simple graphic symbols. Certainly, J. Bertin has studied combinations of retinal variables on them, superposition and containment of points in zones. But complex graphic symbols like UML nodes have not been explicitly explored. The complexity of UML nodes might be observed with one of the most used UML nodes: the UML Class. It consists of a rectangle with four compartments. Each compartment contains text. The contained text must be sufficiently large and easily readable.

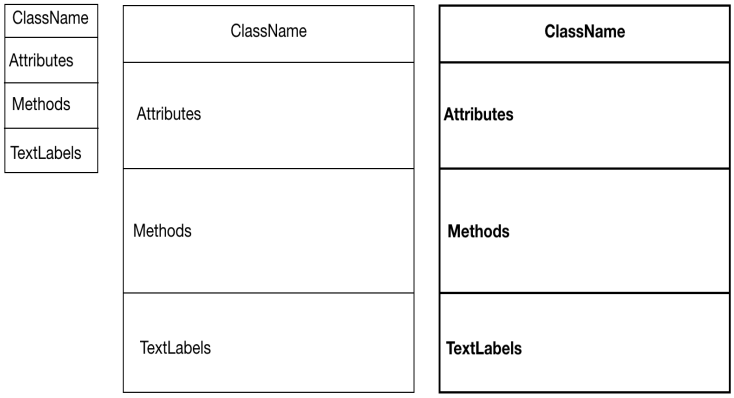

Figure 3.25: Different implementations of the size retinal variable to an UML class.

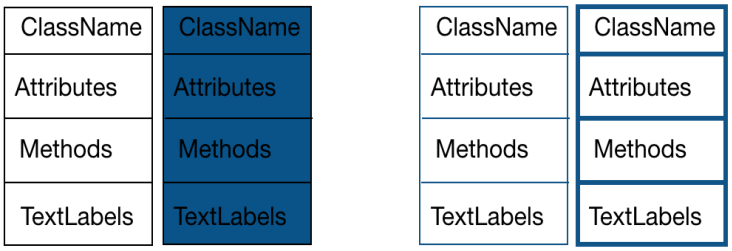

Figure 3.26: Color: different possible implementations on an UML class

In that context, many possible implementations of the retinal variables to the UML nodes exist. For example, the application of the size retinal variable to an UML class might concern only its area. But, it might also concern the area and the contained text of the UML class, as depicted in Figure 3.25 above. Another example concerns the application of a color variation to an UML class. Color can be applied to the background of the class or to its default thin border lines (Figure 3.26). But, the color application might also require increasing the border lines thickness, as shown in Figure 3.26.

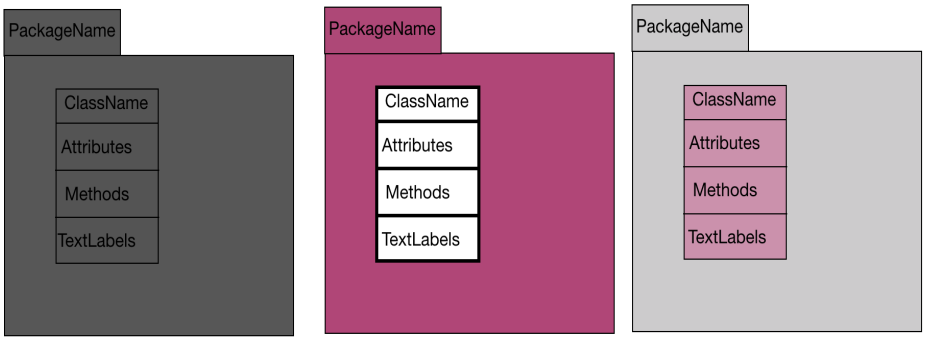

Figure 3.27: Color: taking into account the embedding relationships

Moreover, taking into account the possible embedding of UML nodes when applying a retinal variable to them is problematic. For example, coloring an UML class which is embedded in a colored package might require increasing the thickness of the class border lines (Figure 3.27). Also, the choice of the category of a visual variable for an embedded UML node has to be careful. For instance, a class can have the same color as the package in which it is embedded or a different one (Figure 3.27). Choosing the same color for the embedded class and the package can hamper the readability of the text, especially if the chosen color is dark (Figure 3.27).

UML graphic nodes are then very complex compared to points, lines and zones of the SoG. The SoG principles cannot be applied to UML in an ad-hoc manner. They have to be carefully adapted to the UML graphic complexities and particularities. The graphic complexity of UML graphic nodes has not been addressed neither in the SoG nor in the existing researches.

Reformulation of the research questions

In the first section, we illustrated that the present work is complementary to the CDs framework and to the PoNs framework. Particularly, the work of this thesis is meant to subscribe precise guidelines to reduce the cognitive load in some dimensions of the CDs framework and some principles of the PoNs framework (See Table 3.2), by taking advantage of the retinal variables. To that end, the theories of the SoG were explored in the UML context. But, in the second section, we observed that the rules of the SoG cannot be directly mapped to UML due to the graphic particularities and complexities of the latter modeling language. These findings allow us to reformulate our general research question: *How to improve the effectiveness of UML diagrams via an advanced usage of their secondary notation visual variables?*. The following more precise questions arise:

- **RQ-1:** What are the visual variables which belong to the UML secondary notation?
- **RQ-2:** What are the exact graphic particularities and complexities of UML?
- **RQ-3:** What are the effective ways to better take advantages of them in UML?
- **RQ-4:** How can we take into account the recommendations of the practitioners mentioned in our conducted empirical study?

- **RQ-5:** How can we empirically assess the benefits of the visual variables of the secondary notation in UML?

The next chapters present our contributions which address all of these research questions.

Chapter 4

SoG-UML: Semiological guidelines for the visual enrichment of UML diagrams

Contents

To visually enrich his UML diagrams, a designer cannot use the SoG directly because of its unsuitability for composite graphic elements like UML nodes. Given the eventual interest of such enrichment, we have completed the SoG by specific guidelines for software modeling and particularly for UML. These guidelines are grouped in a framework that we have simply named SoG-UML. To define SoG-UML, we relied on the *design and action* theory, as defined by [26]. A design and action theory is meant to give explicit prescriptions for constructing an artifact (e.g., methods, techniques, principles). We describe then guidelines of effectiveness to better take advantage of each retinal variables in UML. To guarantee a high level of accuracy of our guidelines, we begin by an exhaustive empirical classification of the UML concrete syntax. With such categorization, we became able to identify the retinal variables which belong to the UML secondary notation and to draw an explicit list of all the graphic particularities of UML. As a result, for each graphic element of the list stemming from the previous classification and for each retinal variable of the UML secondary notation we explain explicit guidelines of effectiveness.

4.1 The UML concrete syntax: an exhaustive classification

To analyze the UML graphic space, we started from the UML standard specification [4]. We first classified the UML graphic nodes based on a containment criteria, which refers to the possibility of an UML graphic node to be a container of another one and/or to be contained in another one. Then, we grouped them based on their visual aspects. That denotes their corresponding shapes and characteristics of each shape (i.e., border lines and corners), number of compartments, presence of icons and/or presence of satellites (i.e., ports). We build then a clear vision about the UML graphic space, especially about the visual variables which belong to the UML secondary notation. We also ensure that all the graphic particularities of UML have been identified.

Results of the classification are depicted in the feature models[1] which are shown in Figure 4.1 and Figure 4.2. They are displayed in the next pages for space constraints. As illustrated in the keys of each model, gray rectangles with a thick border denote the visual variables which are locked by the UML primary notation. White rectangles with a thick border denote the main graphic elements which might compose a graphic component (i.e., graphic node or graphic edge). Finally, white rectangles with a thin border refer to the alternatives of a graphic element or a visual variable. The relationships between the different rectangles are equally explained in the keys. They can be mandatory or optional.

Results of the classification show that the UML graphic nodes are complex due to multiple facts. On one hand, UML graphic nodes might be composed of multiple graphic elements. They are mainly characterized by a shape which can be a circle, a rectangle, an ellipse or a symbol (e.g., stick man in a use case diagram). These shapes have borders. Line borders can be continuous or dashed ones (a grain variation). They also might be gray or black to denote a brightness variation (i.e., to differentiate inherited and non inherited states). Corners of the shapes might be curved or folded. UML

[1]Feature model is used to help generating guidelines for other modeling languages as discussed in the perspectives of this thesis.

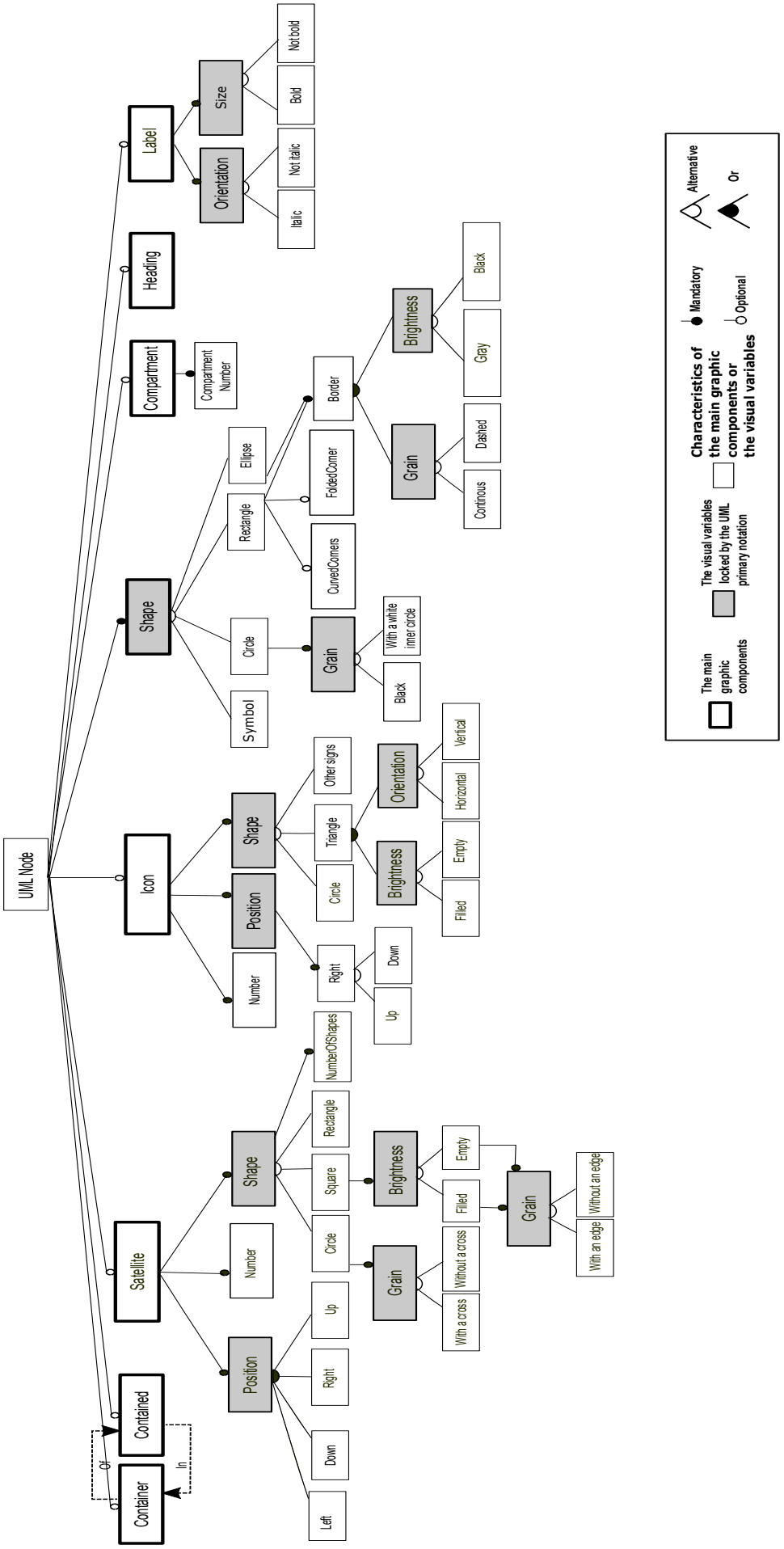

Figure 4.1: Cartography of the UML graphic nodes and the used visual variables.

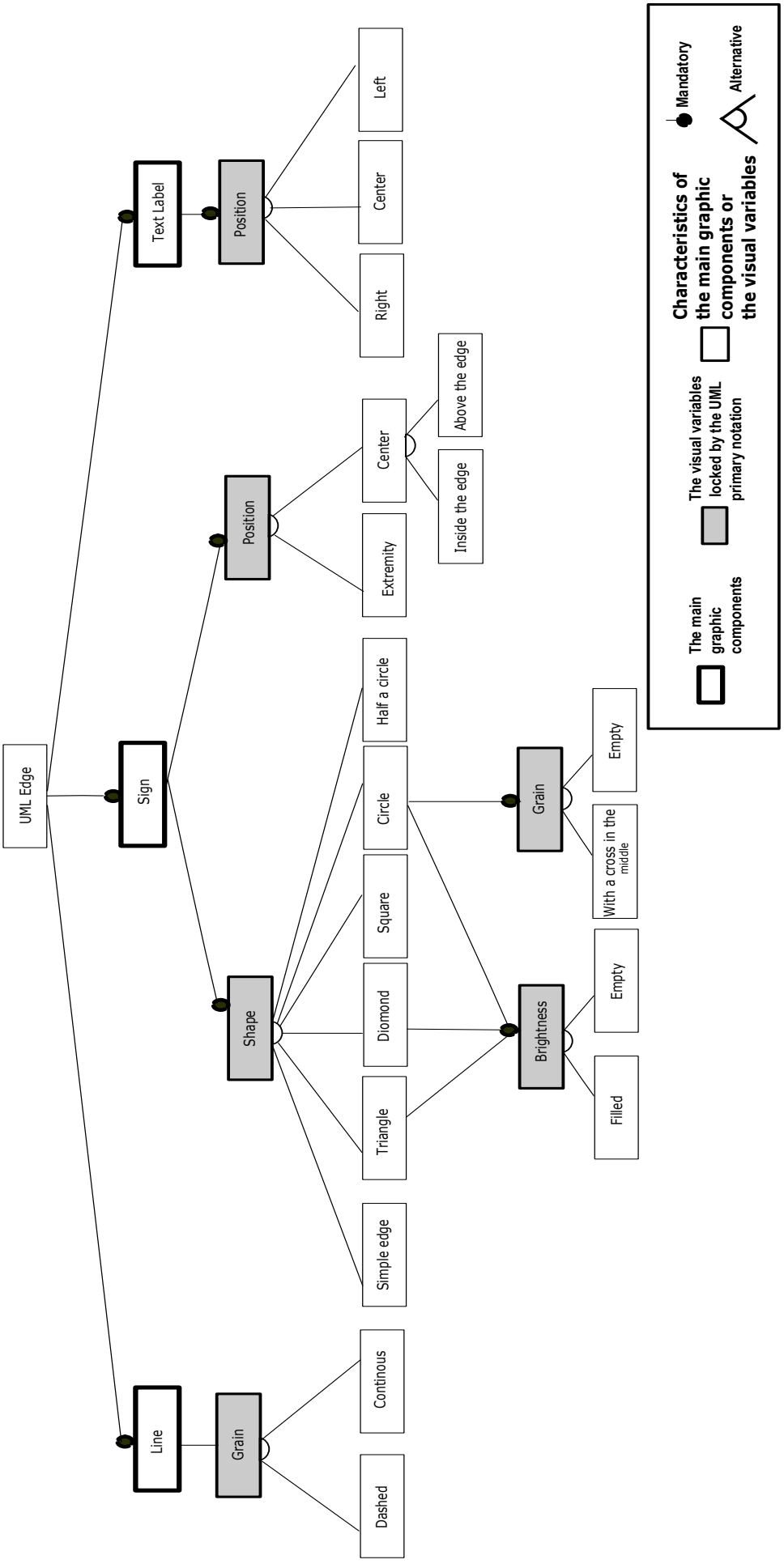

Figure 4.2: Cartography of UML graphic edges and the used visual variables.

nodes might also be composed of one or more than one compartment. In addition, they might contain text which can have an italic font that we consider as an orientation variation (e.g., Text and *Text*). Text labels might also be bold or not which corresponds to a size variation to denote UML active classes (e.g., Text and **Text**). Then, UML nodes might be composed of icons. There may be one or several icons which might be placed in different positions. They are also characterized by their shapes, where we find triangles, circles and other composed ones. Triangles might be empty or filled, which corresponds to a brightness variation. They might be horizontal or vertical, which denotes an orientation variation. Finally, some UML nodes might have attached satellites (i.e., ports). These ones might have different shapes and be characterized by their number and position. Grain and brightness might vary for some satellites. On the other hand, UML nodes are made to be parts of a whole UML diagram. In fact, UML nodes are not only composed of multiple elements (i.e., compartments, text, icons, ports), they can also be totally contained in another graphic nodes (e.g., a class in a package). Therefore, we have two types of UML graphic nodes: containers and contained ones. The Figure 4.3 below illustrates the complexity of UML nodes.

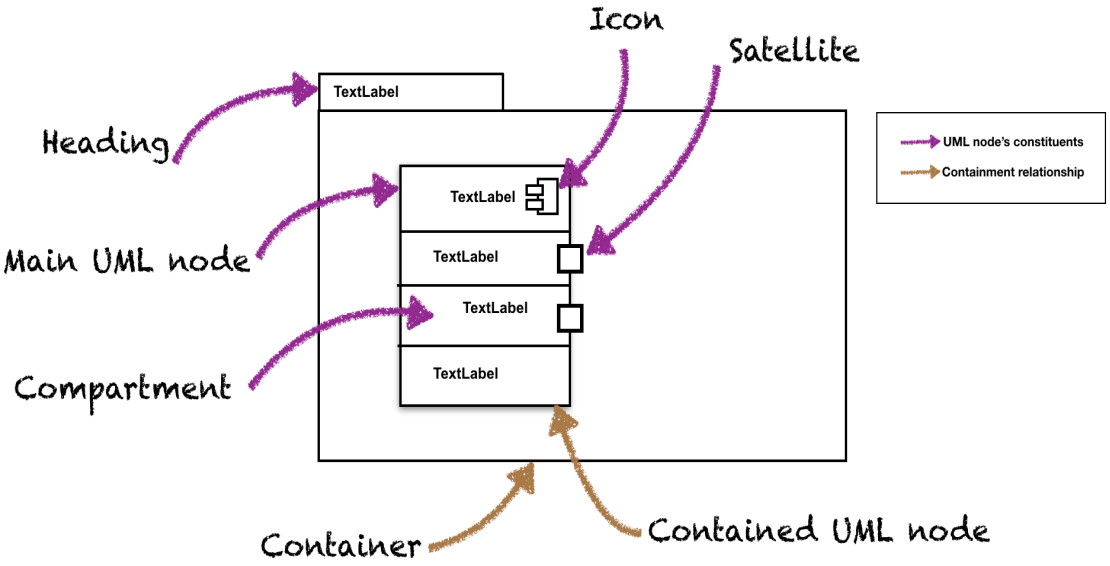

Figure 4.3: The identified graphic elements.

Concerning UML edges, Figure 4.2 illustrates all the constituents of an UML edge and highlights the visual variables which are blocked by the UML primary notation. UML edges are mainly composed of a line which might be continuous or dashed (a grain variation). They are also characterized by a sign which has a particular shape and position and also by a text label that has a fix position.

4.2 SoG-UML

We studied in SoG-UML all the retinal variables of the UML secondary notation based on the previous classification. We excluded the position because we find that there exist a huge effort in the literature that try to find the effective layouts (e.g., [45] [40] [37]). For each retinal variable, we show and justify how to implement it as efficiently as possible on all the groups stemming from the previous classification (i.e., main UML

node, icons, ports, text labels, headings and compartments). For that, we pragmatically expose the possible implementations and we identify the choice of the most effective one based on the SoG. The rules of the SoG that have been used to justify it are summarized in a table. We define for each rule in the table a *rule identifier*, *a title*, *the rule* itself and its *explanation*. We distinguish two types of rules: basic rules like definitions or rules that concern any visual variable and specific rules that concern one particular visual variable. Basic rules have a *rule identifier* that begins with the letter *B*. The other rules have a *rule identifier* that begins with the first letter of the corresponding visual variable (e.g., C for color, S for size). *BR* is used as a rule identifier for the brightness visual variable (i.e., to avoid using the letter B which is used for basic rules). We mention the text from the book of the SoG that proves each rule in the explanation column. That text is the result of our own translation from French to English and is preceded by the page number in the book in the form of [*SoG-Page Number*]. We generate then a set of guidelines for each group. They are in the form of *Should* and *Should not* and are put in bold after their explanation.

An illustration scenario: Internet Banking System We will illustrate SoG-UML along with an example of a scenario inspired from the results of our previous empirical study (See Chapter 2). The scenario is based on a real project found in the Lindholmen GitHub models repository [3]: Internet Banking System (IBS). In the

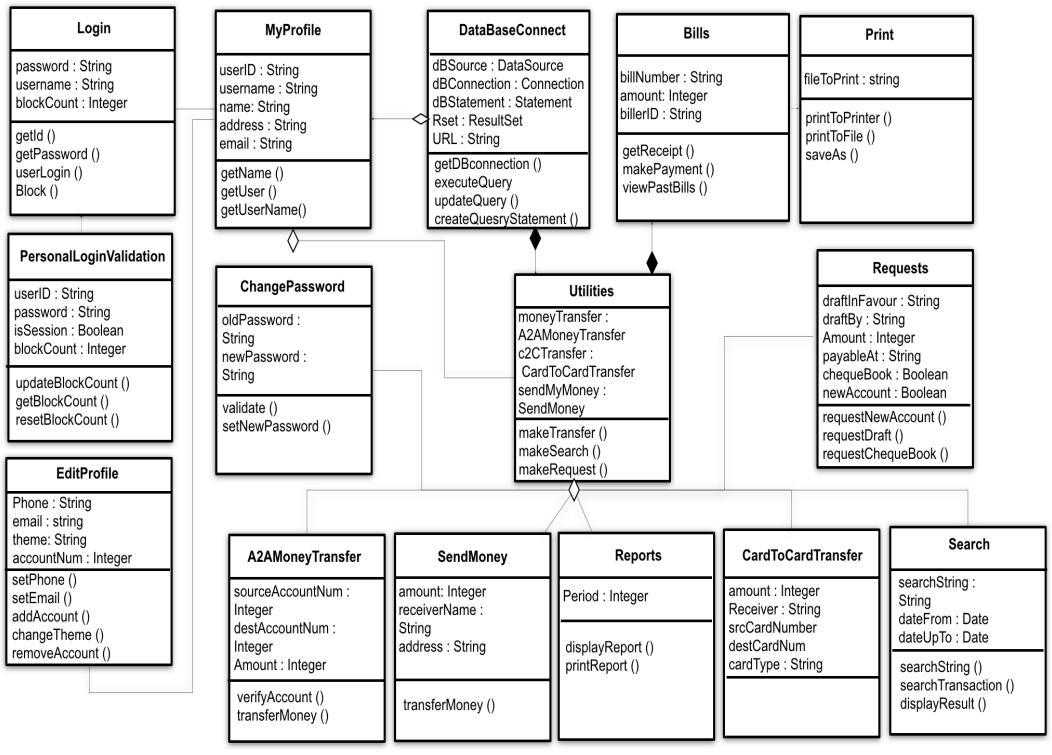

Figure 4.4: The class diagram of the Internet Banking System created by its developers.

context of a competition organized by IBM called The Great Mind Challenge [28], a group of developers participated with the IBS project and got the 9th position. Figure 4.4 illustrates their class diagram. IBS is an online service which allows users to log in, using a personal identifier and a password, into their banking account profile. The classes which are responsible of the log in functionality are: **Login** and **Personal-LoginValidation**, as shown in Figure 4.4. Users can view information related to their

accounts and edit them via the classes: **MyProfile, ChangePassword** and **EditProfile**. They are able to generate reports via the class named **Report**, search for data about a particular period of time via the class named **Search**. They can also extract bills and print them. The two following classes are responsible of these features: **Bills** and **Print**. They can also request for Cheque books via (**Request**) and make different online fund transfers (**A2AMoneyTransfer, SendMoney, CardToCardTransfer**). Finally, all the data of the IBS are centralized in a database: **DataBaseConnect**.
We will use the visual variables to highlight information which might be relevant for communications with the competition committee and so related to *System's concerns, distribution of tasks, project progression, criticality and correlations between these latter.* The diagram in Figure 4.4 is used as a reference in the forthcoming subsections.

4.3 SoG-UML: Color

Table 4.1 summarizes the rules of the SoG that we use to generate our guidelines about the effective use of colors in UML.

Table 4.1: The used rules from the SoG to generate our guidelines about colors in UML.

Rule identifier	Title	Rule	Explanation
B1	Definition	**The color variation is the differentiation generally induced by different colored excitation that the human eye can perceive between two ranges having *the same brightness***	Color and brightness are two different retinal variable. Each one has its particular perceptive properties. [**SoG-85**]: *We cannot understand the colors sensations if, first of all, we do not separate rigorously the notion of colors of the notion of brightness.*
B2	One-to-one mapping of information properties	**In a diagram, colors should express one single information property.**	During the analysis process, each information property should be mapped to one single retinal variable [2] which has (at least) the same organization level. [**SoG-10**] *It is important to transcribe each information property by one visual variable which has at least the corresponding organization level and the length.*
C1	Effective categories of colors	**Medium levels of brightness provide the biggest number of selective colors.**	The medium level of brightness offers seven selective colors. While the low level provides four selective colors and the high level gives only three selective colors. [**SoG-87**] *Colors in medium level of brightness offer the biggest number of selective colors.*
C2	Color and size	**The smaller the spot is, the more important the time necessary to distinguish the used colors is and vice versa.**	Big colored spots are easily distinguishable by the human eye compared to smaller ones. [**SoG-89**] *The smaller the spot is, the less the colors are visually distinguishable.*
C3	Length of the color variation and size	**The smaller the spot is, the less the number of selective colors is and vice versa.**	The length of the color variation depends on the size of the spot. [**SoG-89**] *For very big areas, the number of selective colors might reach one million colors.*

Table 4.1 continued from previous page

C4	Colors and the types of the spots	**In lines or points, light colors are excluded because they are very near to white, the default background color.**	The selectivity of colors depends on the type of the spot. The visual selection of light lines and points against a white background is very difficult. This fact is due to the very low contrast between white and light colors. **[SoG-89]** *In lines (or points), light colors are excluded. Black can replace them and the ranges of colors will have a great visibility.*

4.3.1 Color variation on the main UML node

We begin by studying the effective implementation of color on the main UML node, here an UML class. For that, the Figure 4.5 illustrates the possible implementations.

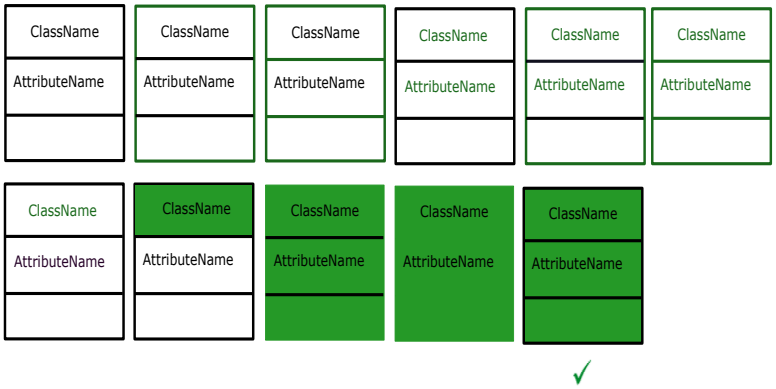

Figure 4.5: The possible implementations of the color retinal variable on an UML class.

The selectivity of color increases when the size of the corresponding spot raises (**C2** - *Color and size*). It is likewise for the number of selective colors. The bigger the spot is, the higher the number of selective colors is (**C3** - *Length of the color variation and size*). In addition, light lines cannot be accurately perceived on a white background (**C4** - *Colors and the types of the spots*). In that context, as border lines, lines of separation and text are relatively thin, if we apply the color on them, such implementation will negatively impact the selectivity of colors, the number of selective colors and their visibility when light colors are employed. The reader can observe this in the Figure 4.6 below.

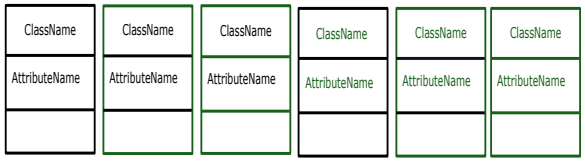

Figure 4.6: Colors and the borders, separation lines and text thinness.

Then, the following guideline stems:

> **[SoG-UML #1] Colors should not be applied to borders, lines of separation and/or text, when they are relatively thin.**

Now, we explain the corollary statement to the previously defined guideline. Based on **C2** - *Color and size* and **C3** - *Length of the color variation and size*, for an application on the borders, separation lines and text: the bigger their size/thickness is, the better the selectivity of colors on them is and the better light colors are selected against the white background (**C4** - *Colors and the type of the spots*). Figure 4.7 below illustrates this reasoning on an UML class. So, based on this argumentation, we generate the following guideline:

> **[SoG-UML #2] The thicker the border and separation lines are, the better colors are visually selected.**

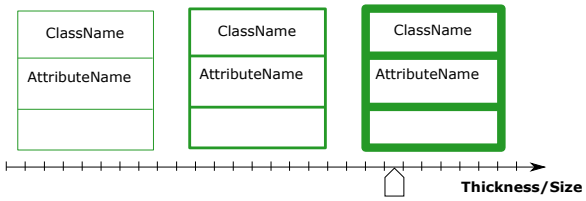

Figure 4.7: Impact of the line thickness on the selectivity of colors.

In order to fully benefit from the color variation in terms of selectivity (**C2** - *Color and size*) and number of selective colors (**C3** - *Length of the color variation and size*), colors should be applied to a big area of an UML node. For that, an alternative of implementation is available. It consists of coloring the UML node's name compartment, as illustrated in Figure 4.8.

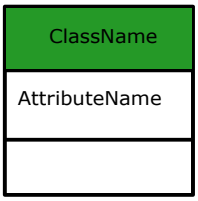

Figure 4.8: Coloring the UML node's name compartment.

But, coloring all of the compartments produces a more effective implementation because it concerns the biggest area of an UML node, as required by the rules **C2** - *Color and size* and **C3** - *Length of the color variation and size*. The following guideline arises:

> **[SoG-UML #3] Colors should be applied to the background of an UML node.**

To apply a color variation to the background, two possibilities exist as shown in Figure 4.9 below. We observe that these implementations are susceptible to change the shape of the concerned UML node and so might alter the UML primary notation: an UML class can no more be perceived as a rectangle with X compartments. Figure 4.10 below illustrates then the effective implementation of the color retinal variable on an UML class. The same type of implementation is appropriate for UML Decision, Use Case and activity, as illustrated in Figure 4.11.

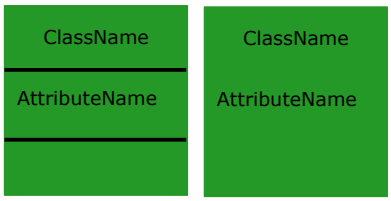

Figure 4.9: Implementations of colors which might alter the UML primary notation.

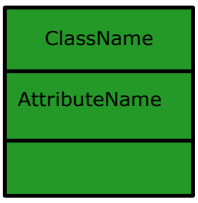

Figure 4.10: Effective implementation of the color retinal variable on an UML class.

Figure 4.11: Effective implementations of the color retinal variable to 4 different UML nodes.

4.3.2 Color variation on text labels, borders and separation lines

We have seen that coloring the background of an UML node is the best support to implement the color variation. Now, we have to identify the set of colors which yield to the best selectivity, but more important, we need to state the colors of text and borders to use regarding the background.

Based on **C1** - *Effective categories of colors*, medium level of brightness for colors is the best level because it provides the biggest number of selective colors (High, medium and small levels of brightness were explained in *Chapter 3*). In UML, high levels of brightness ensure the best readability for black text labels and the best visibility of the borders and separation lines. The contrast is sufficiently big to see the difference between a bright background and black lines (**C4** - *Colors and the type of the spots*). However, this level provides only three selective colors. Whereas, the medium level gives seven selective colors while ensuring the readability of black text and the visibility of the other black constituents of an UML node (i.e., borders and separation lines). Hence, the three following guidelines result:

> **[SoG-UML #4] Colors in medium levels of brightness should be used for UML nodes. They ensure the readability of black text labels, borders and separation lines and provide the biggest number of selective colors.**

> **[SoG-UML #5] In medium levels of brightness, text labels, borders and separation lines should be black.**

> **[SoG-UML #6] In high levels of brightness, text labels, borders and separation lines should be black.**

Figure 4.12 below illustrates the effective implementation of colors in medium level of brightness in the IBS class diagram.

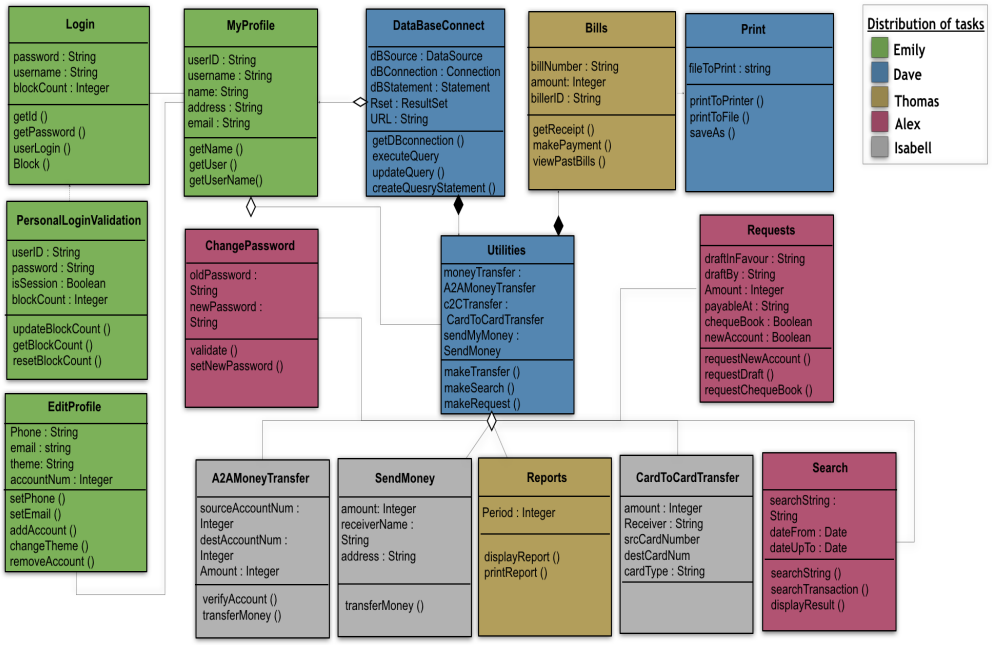

Figure 4.12: IBS class diagram: Medium level of the background color's brightness.

Colors are used to express the distribution of tasks between the developers of the system: *Emily, Dave, Thomas, Alex and Isabell*. The medium level of brightness allowed us to employ five selective categories of colors. They are shown in Figure 4.13 with their RGB (Red, Green, Blue) and HSB (Hue, Saturation, Brightness) codes.

RGB: (124, 178, 91)	RGB: (178, 86, 118)	RGB: (86, 134, 178)	RGB: (178, 178, 178)	RGB: (178, 157, 92)
HSB: (97, 49, 70)	HSB: (339, 52, 70)	HSB: (209, 52, 70)	HSB: (209, 0, 70)	HSB: (45, 48, 70)

Figure 4.13: RGB and HSB codes of the colors used in Figure 4.12.

If we want to express the same information (i.e., the distribution of tasks) using colors in high levels of brightness, we will not be able to represent all of its five categories. In fact, this level provides us with only three selective colors. Figure 4.15 below shows how we used this level to represent the distribution of tasks of only three developers in the IBS class diagram. RGB and HSB codes are also detailed in Figure 4.14.

RGB: (229, 254, 206)	RGB: (255, 253, 218)	RGB: (254, 232, 204)
HSB: (91, 19, 100)	HSB: (57, 15, 100)	HSB: (34, 20, 100)

Figure 4.14: RGB and HSB codes of the colors used in Figure 4.15.

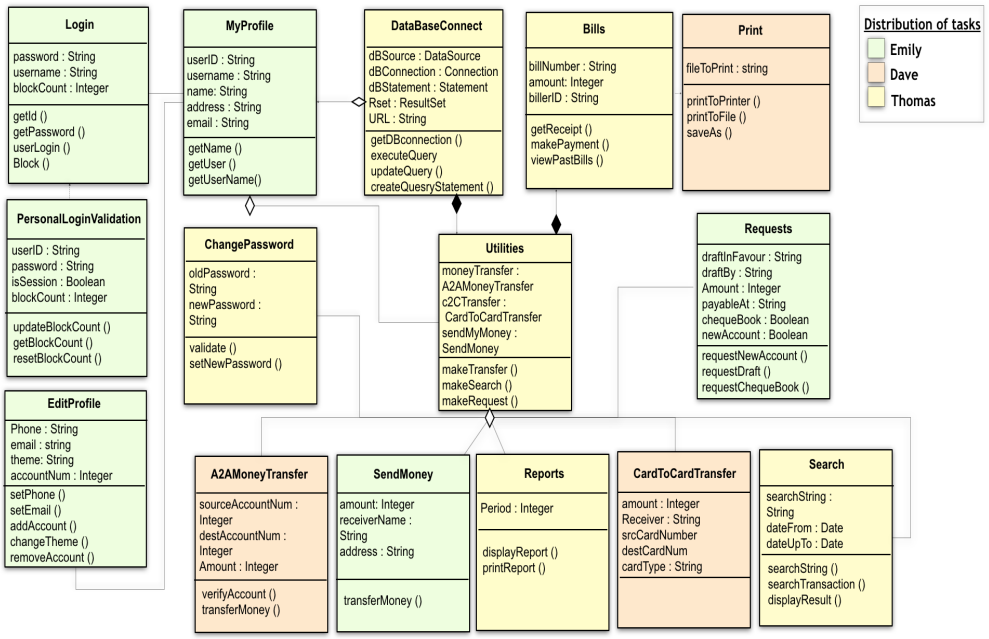

Figure 4.15: IBS class diagram: High level of the background color's brightness.

Now, if a designer wants to use dark colors to express a particular information on his diagram, keeping the default black color of text labels, borders and separation lines represents a big mistake as illustrated in Figure 4.16 .

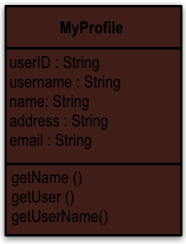

Figure 4.16: Unreadability of the text labels in a dark background color of an UML node.

C4 - *Colors and the type of the spots* concerns the selectivity of lines on white backgrounds where black ones should be used to increase the contrast between both of them. In this case, we discuss the selectivity of lines on dark backgrounds (i.e., the background colors of the UML node). So, white text labels, borders and separation lines should be used to ensure a maximum contrast. From where the following guideline is generated:

> **[SoG-UML #7] In low levels of brightness, text labels, borders and separation lines should be white.**

Figure 4.17 below illustrates the effective implementation of text labels, borders and separation lines on dark background colors of UML nodes, in the class diagram related to the IBS. Dark colors are employed to highlight the main concerns of the IBS: *Profile management, Database management and Utilities* and Figure 4.18 shows the RGB and the HSB codes of the used colors.

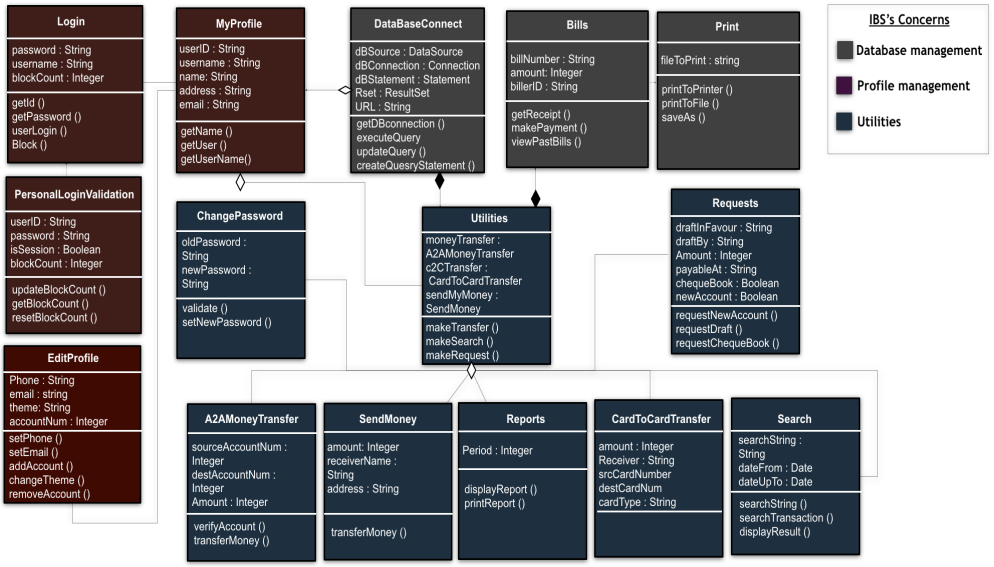

Figure 4.17: IBS class diagram: Low level of the background color's brightness.

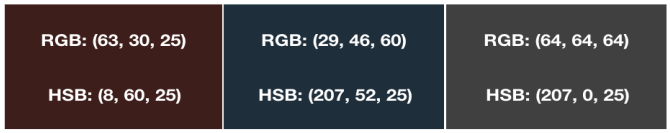

Figure 4.18: RGB and HSB codes of the colors used in Figure 4.17.

In that context, note that:

> **[SoG-UML #8] In medium brightness we find the maximum number of selective colors. The blue and red colors are diametrically opposite: violet, blue, green, yellow, orange, red and purple might be used in this level of brightness.**

> **[SoG-UML #9] In big brightness: colors are chosen around the yellow, from green to orange. Light blue, violet, purple and red are less selective**

> **[SoG-UML #10] In low brightness (i.e., dark colors): the selective categories of colors range from blue to red by the violet and purple. Dark green, yellow and orange are less differentiated.**

4.3.3 Color variation and icons

We showed in the previous paragraph that the readability of the text labels, the borders and the separation lines depends on the background color of their UML node. Icons rely on the same parameter (i.e., the background color of its UML node) because they are also constituents of UML nodes. But, icons are characterized by two graphic elements: borders and a background. To ensure their selectivity against the background of their corresponding UML node, we will mainly rely on their background which represents their biggest area. In fact, based on **C2** - *Color and size*, the bigger the spot is, the better colors are visually selected, and, in our case the better icons are selected from the main UML node. For that, icons borders might keep their default black color.

But, they should have a different background color from their owning UML node. For example, the icon is better visually selected when it has a gray background color than white. Figure 4.21 below shows this implementation on a component diagram of the IBS.

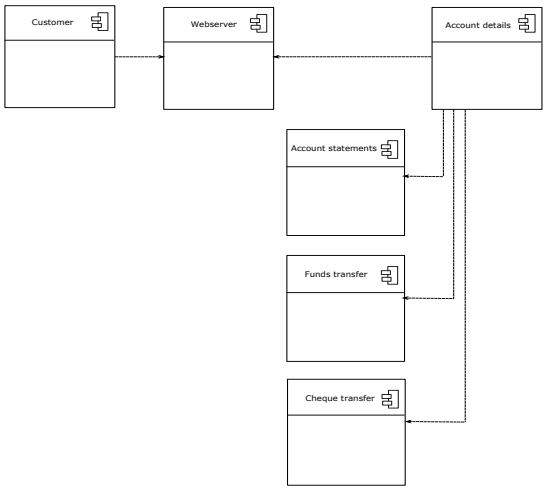

Figure 4.19: The same background color for UML nodes and icons.

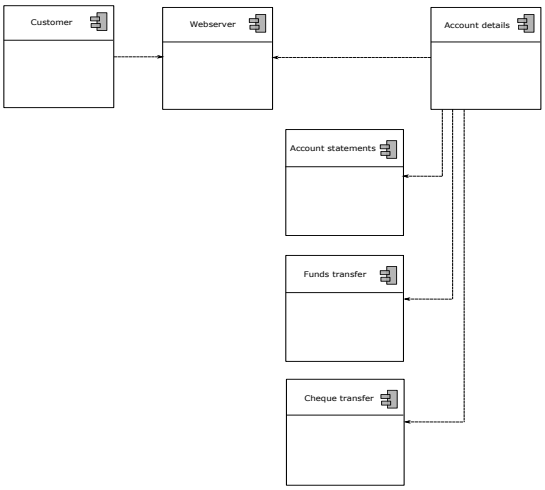

Figure 4.20: Different background colors for UML nodes and icons.

Figure 4.21: Effective implementation of an icon in a white UML node.

We conclude then the following guidelines:

> **[SoG-UML #11] The border of icons should be black.**

> **[SoG-UML #12] Icons should not have the same background color as the UML node in which they are contained.**

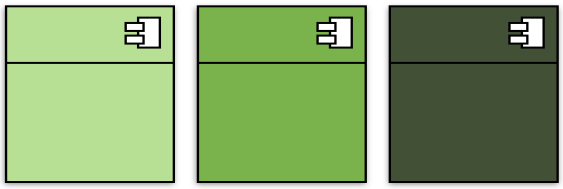

Figure 4.22: White background color of icons in all levels of brightness of the main UML node's background color.

Concerning colored UML nodes (neither white nor black), white background of icons might be suitable for all colors in all the three possible levels of brightness. In fact, for colors in big and medium levels, black borders of icons help to rapidly see them. For low levels (i.e., dark colors), the white background of icons is excellent to rapidly perceive them (i.e., thanks to the big contrast), as shown in Figure 4.22 above. Figure 4.23 below illustrates the effective implementation of icons which are contained in colored UML nodes.

But, it is important to notice that icons should not have different colors from each other. As stated by **B2** - *One-to-one mapping of information properties*, colors

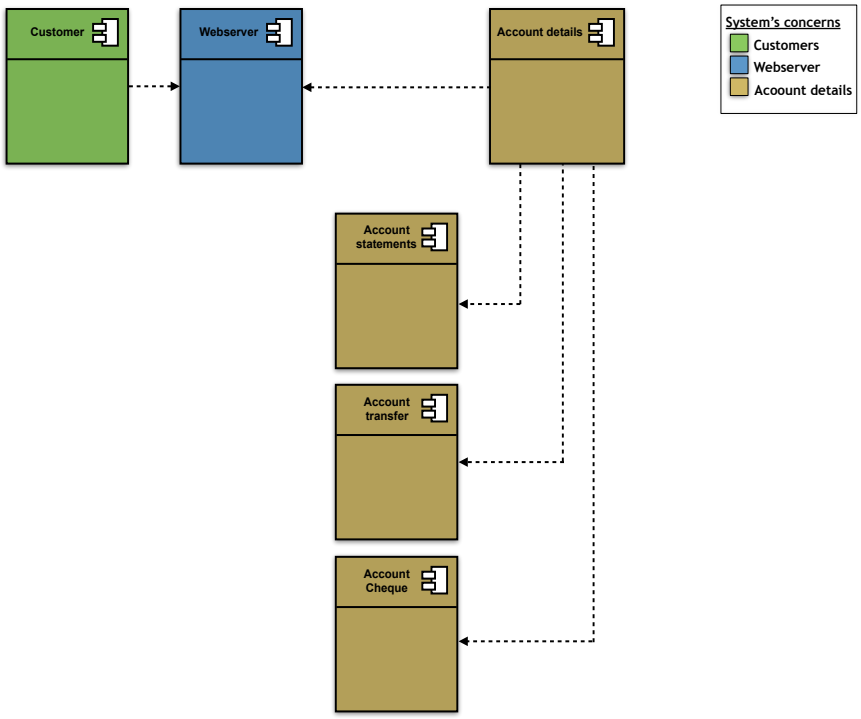

Figure 4.23: Effective implementation of icons in colored UML nodes.

have to be employed to express only one information property. If they are used to convey a particular information property via the main UML node's backgrounds, a color variation on icons will induce to ambiguities. Hence, we produce the following guideline.

> **[SoG-UML #13] All of the icons in a diagram should have the same background color.**

4.3.4 Color variation on UML satellites/ports

To simplify the explanation, we propose to name the UML node in which the port is attached as ported UML node. The color of ports depends on two parameters: the color of the ported UML node and the color of the ported UML node's container. To increase the selectivity between these three graphic nodes, colors should be applied to the port's biggest area which means its background, as explained by **C2** - *Color and size* and **C3** - *Length of the color variation and size*. The following guideline is then prescribed:

> **[SoG-UML #14] Colors should be applied to the background of ports.**

Concerning the ports borders, as for icons, black ones are suitable for all levels of brightness of the used colors. The following guideline results:

> **[SoG-UML #15] The border of ports should be black.**

Now, a ported and white contained UML node might be white or colored, as shown in the Figures 4.24 and 4.25. In both cases, the port should have a different selective

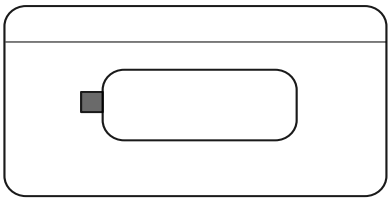

Figure 4.24: Effective implementation of a port in a white ported and white contained UML node.

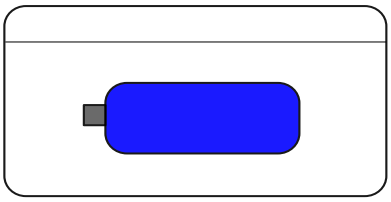

Figure 4.25: Effective implementation of a port colored and white contained ported UML node.

Figure 4.26: Effective implementation of a port in a white container.

color from it. But, in the case of a colored container, the ported UML node might also be white or colored, as illustrated in the Figures 4.27 and 4.28. If this latter is white, the port should have a different background color from the container. Otherwise, the port should have a different selective color from the container and the ported UML node.

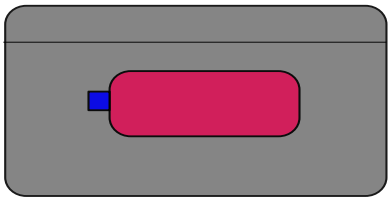

Figure 4.27: Effective implementation of a port in a colored ported UML node and contained in a colored container.

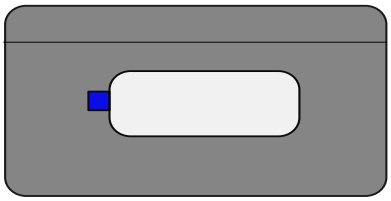

Figure 4.28: Effective implementation of a port in a white ported UML node and contained in a colored container.

The following guideline is then prescribed:

> **[SoG-UML #16] Ports should have a different selective color from the UML node in which it is embedded and the container of this latter.**

4.4 SoG-UML: Brightness

Table 4.2 contains the employed rules of the SoG which helped use to generate our guidelines about the effective use of the brightness in UML.

Table 4.2: The used rules from the SoG to generate our guidelines about the brightness in UML.

Rule identifier	Title	Rule	Explanation
B3	Definition	**The brightness variation is the continuous progression that the eye perceives in the series of gray, which ranges from black to white.**	This variation is ordered, from the black which has the lowest brightness to white which has the biggest one. **[SoG-73]**: *This progression is independent from the color and we can range from black to white by gray, blue, red, etc.*

Table 4.2 continued from previous page

BR-1	High brightness and selectivity	**In big levels of brightness, the number of selective categories of all the retinal variables, including shapes, decreases to reach zero with null brightness (i.e., white spots).**	Designers should be careful about the use of high levels of brightness because it hampers the selectivity of the other retinal variables. [**SoG-73**]: *In very big values of brightness, the number of categories of sizes, colors, shapes, orientations and grain decreases which can be differentiated decreases to reach zero with the null brightness.*
BR-2	Brightness and size	**The smaller the spot is, the smaller the number of selective categories of brightness is.**	The small size of spots negatively impacts the perceptive properties of the used retinal variables on them including the brightness. [**SoG-73**]: *The length of the brightness varies with the size of the spots. The more the spot is small, more the length of selective categories are reduced.*
BR-3	Brightness and dissociation	**The brightness is dissociative. It prohibits the visualization of spots, all categories combined.**	Models authors should be careful about the use of the brightness because it is dissociative (It means that it hinders the visualization of the UML nodes as a group.). [**SoG-73**]: *The brightness is dissociative and it is impossible to visually abstract it.*
BR-4	Effective categories of brightness	**To have the biggest number of effective categories of brightness, black and white should be used.**	For three effective categories of brightness, choose: black, white and gray. For four categories, choose black, white and two levels of gray. [**SoG-73**]: *The length of the brightness depends evidently on available distance between the black and white.*

4.4.1 Brightness variation on the main UML node

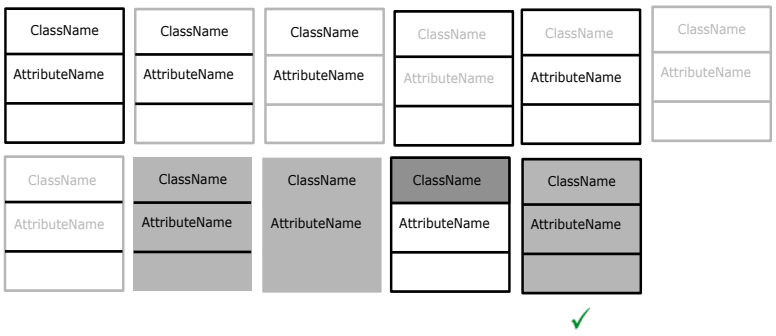

Figure 4.29: Possible implementations of the brightness retinal variable on an UML class.

Figure 4.29 above shows the possible implementations of the brightness retinal variable on an UML class.
The brightness is defined as the continuous progression that the eye perceives from black to white via a series of gray or another color (**B3** - *Definition*). In that context, a small size of the spot decreases the number of selective categories of brightness (**BR-2** - *Brightness and size*). In addition, high levels of brightness hinder the selectivity of the other retinal variables, including the shape which is a key visual variation in UML (**BR-1** - *High brightness and selectivity*). Based on these rules, borders, text and separation lines are relatively thin, they cannot effectively support a brightness variation,

as depicted in Figure 4.30.

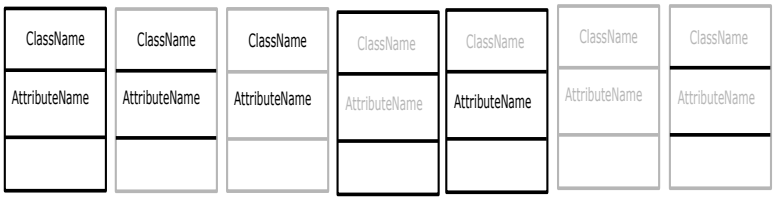

Figure 4.30: Brightness and borders, separation lines and text thinness.

Besides, this kind of variation is already employed by the UML primary notation to differentiate between inherited and non inherited states, which is not effective as argued earlier. We conclude then the following guideline:

> **[SoG-UML #17] The brightness variation should not be applied to the borders, separation lines and text: it is locked by the UML primary notation and not effective from the SoG point of view.**

To take advantage of the whole range of selective categories of brightness, we have to implement it to a relatively big area of an UML node (**BR-2** - *Brightness and size*). The brightness variation might be applied to the class name compartment. The area is relatively big compared to the borders, separation lines and text labels. Such implementation allows a better perception of the brightness variation, as shown in Figure 4.31.

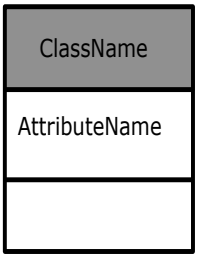

Figure 4.31: Applying the brightness to the UML node's name compartment.

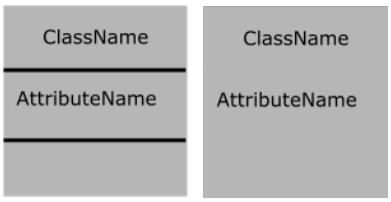

Figure 4.32: Implementations of brightness which can alter the UML primary notation

Obviously, the biggest part of an UML node is its background. Figure 4.32 illustrates the two first possibilities of applying the brightness to the background of UML nodes. But, as for colors, these implementations might alter the shape of the concerned UML node where an UML class is no more perceived as a rectangle with X compartments.

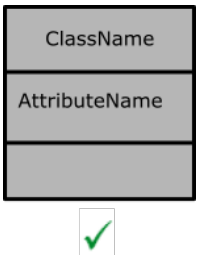

Figure 4.33: Applying the brightness to only background

Figure 4.33 shows a more effective implementation where the brightness is applied to the background of the UML node. Such implementation allows to fully exploit the perceptive properties of the brightness retinal variable in UML. It fits also with Decision, Use Cases and activities, as shown in Figure 4.34 below.

Figure 4.34: Effective implementation of the brightness to different UML nodes

We conclude then the next guideline:

> **[SoG-UML #18] The brightness should be applied to the background of UML nodes.**

4.4.2 Brightness variation and text, separation lines and borders

We have deduced that the brightness should be applied to the background of UML nodes. But, this choice might hamper the readability of the contained text as well as no guidelines precising how the text brightness should be adapted to such variation. In that context, in high brightness the text might be black, which represents the biggest contrast that can exist between both graphic elements. In low brightness (i.e., dark backgrounds of UML nodes), the text brightness has to change to ensure its readability. In a diagram, a first possible implementation consists at using white text for all UML nodes having dark brightness and the default black text for the other nodes, as shown in Figure 4.35. However, as explained by **BR-3** - *Brightness and dissociation*, the brightness is dissociative. Hence, the white text labels and the black ones will be perceived as two different groups (i.e., dissociated). This phenomena is illustrated in Figure 4.35 below where the brightness variation is used to express the ordered information *progress of the implementation* of the IBS project. On one hand, the text in dark gray classes is white. On the other hand, the text of the two other categories of brightness is black. Such implementation might create an ambiguity to the human readers. In fact, an additional non-wanted brightness variation is created via two groups of white and black texts. Moreover, classes having black text are grouped into one group while belonging to two different categories of progression (i.e., *60% and*

30% progress), each one is expressed by a level of brightness.

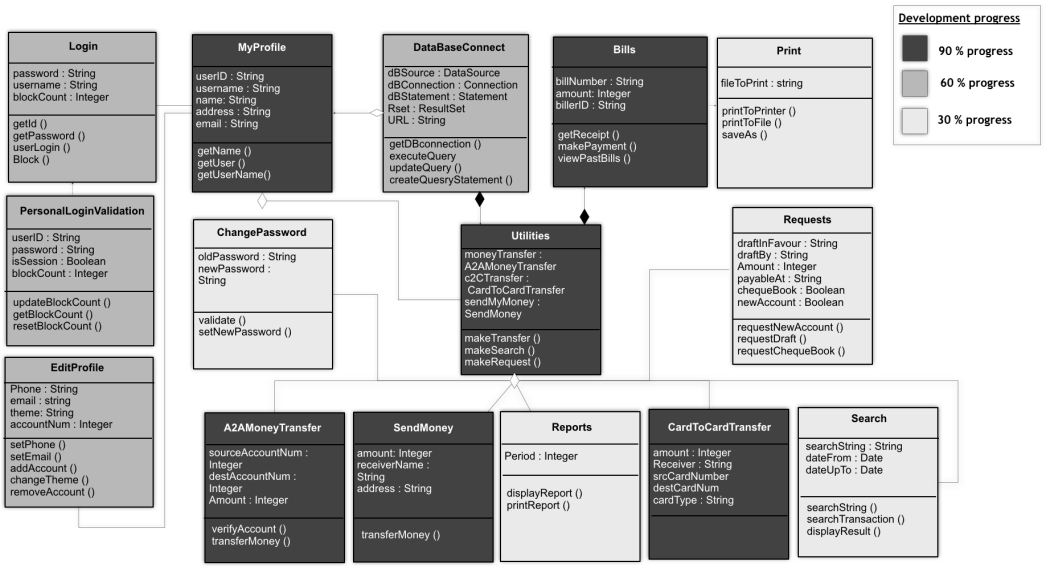

Figure 4.35: Text labels should not have different levels of brightness.

The following guideline comes then:

> **[SoG-UML #19] Text labels, separation lines and borders should not have different values of brightness from each other.**

A solution could be the following: eliminate low levels of brightness which oblige the models authors to change the text brightness to white, as illustrated in Figure 4.36 below. The latter decision will negatively impact the number of categories that can be used as stated by **BR-3** - *Brightness and dissociation*. However, it is the most suitable one in UML.

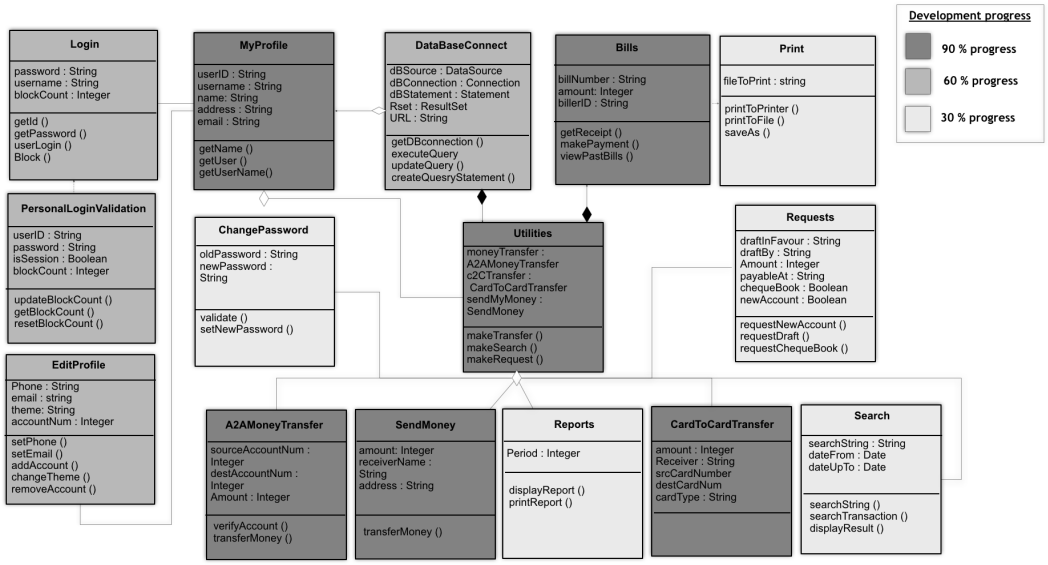

Figure 4.36: The effective implementation of the brightness retinal variable in the IBS class diagram.

4.4.3 Brightness variation and icons

As depicted in Figure 4.1, the brightness of icons is locked by the UML primary notation. It has not to be changed because it might alter their semantic signification.

> **[SoG-UML #20] The brightness of icons is locked by the UML primary notation.**

4.4.4 Brightness variation on ports/satellites

The brightness of the ports is also locked by the UML primary notation. They might be black or white. We precise then the following guideline:

> **[SoG-UML #21] The brightness of ports is locked by the UML primary notation.**

4.5 SoG-UML: Size

Table 4.3 outlines the rules of the SoG which has been used to argue about the effective use of the size variation in UML.

Table 4.3: The used rules from the SoG to generate our guidelines about the size variation in UML.

Rule identifier	Title	Rule	Explanation
B6	Definition	**The variation of the area of an UML node represents the stimuli of the size variation.**	In the quantitative perception, the areas should be proportional to the quantities it represents. [**SoG-71**]: *Any spot, having a punctual or a linear signification, can vary its size without changing its position, brightness, grain, color, orientation or shape. The variation of the area of a spot represents the visual stimuli to the size variation.*
S1	Size and brightness	**The bigger the brightness is, the more the perceptive properties of the size variation are loosed.**	The size variation on empty and white spots is not rapidly perceived by the human reader. [**SoG-71**]: *The size variation is perceptible only with spots having dark values of brightness. Empty and white spots make the size variation lose all its spontaneous perceptive properties.*
S2	Size in points and zones	**Contrarily to points, zones cannot change their size (area).**	The constituents of a zone can vary their size (e.g., line thickness). [**SoG-71**]: *Zones cannot change their area. But their constituents, points or lines can vary in number or size. The extension of the size variation is limited by the concerned zone.*
S3	Size and dissociation	**The size retinal variable is dissociative.**	It prohibits the visualization of the spots as one entity. [**SoG-323**]: *It (the size variation) excludes the perception of the density of a group all categories combined*

Table 4.3 continued from previous page

S4	Size and combinations of forms	**Combining points, lines and/or zone in a graphical representation increases their visual selectivity.**	[**SoG-189**]: *Using different forms of spots (i.e., points, lines and zones) guarantees a high selectivity. It is easy to use and very effective. It is the most effective formula recommended for all sketching/drawings*

Figure 4.37 below shows the possible implementations of the size retinal variable to an UML node.

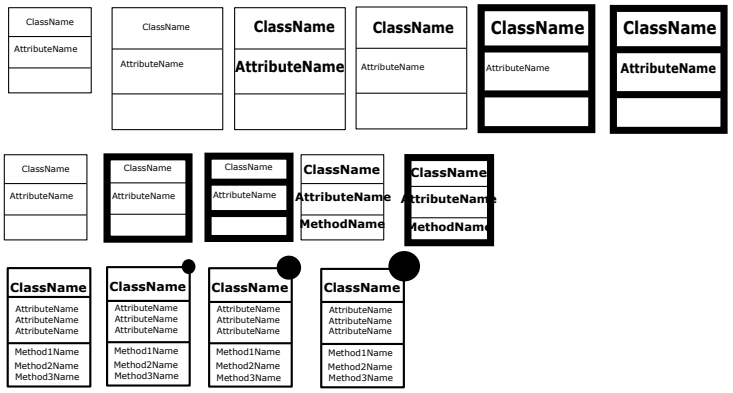

Figure 4.37: Possible implementations of the size retinal variable in a UML class.

As our guidelines are based on the SoG, we propose to build them on the same hypothesis. We consider then that the whole diagram might be visible in the same view (i.e., no hidden parts). That means that it might be seen "at a glance". In such context, we observe that the size variation depends on multiple factors. First, UML diagrams might have a big visual density (i.e., contain a lot of UML graphic components). Varying the area of each UML node will affect the size of the whole diagram. That might prohibit the visualization of the whole diagram at a glance. Our choice of effective implementations takes into account this fact. In addition, the default area of UML nodes depends on their contained text. Generally, UML practitioners use long names of UML graphic nodes (e.g., classes, components, lifelines). Also, UML nodes contain different amounts of text (i.e., number of attributes and methods). In that context, we propose that all UML nodes, contained in a diagram, have the same default area at the beginning of the application of the size retinal variable to an UML diagram. Below is our reasoning about the effective implementations of the size retinal variables.

4.5.1 A- UML diagrams that support a size variation

Size variation on the main UML node

Based on **B6** - *Definition*, it is the variation of the area of a spot which represents the stimuli of the size variation. The following guideline is then generated:

> **[SoG-UML #22] The size retinal variable should be applied to the area of an UML node.**

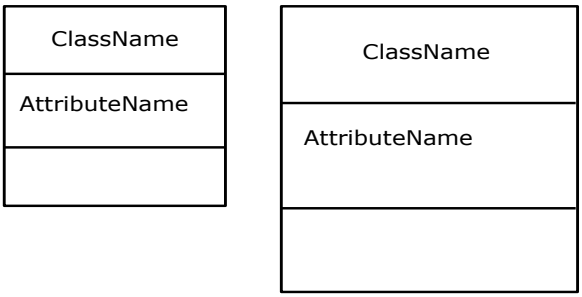

Figure 4.38: Size variation on the area of UML nodes.

Figure 4.38 above shows how we vary the size of an UML node by increasing its area via its height and width.

Figure 4.39 below illustrates the same variation in the IBS class diagram. The size retinal variable is used to express the information property *Criticality*.

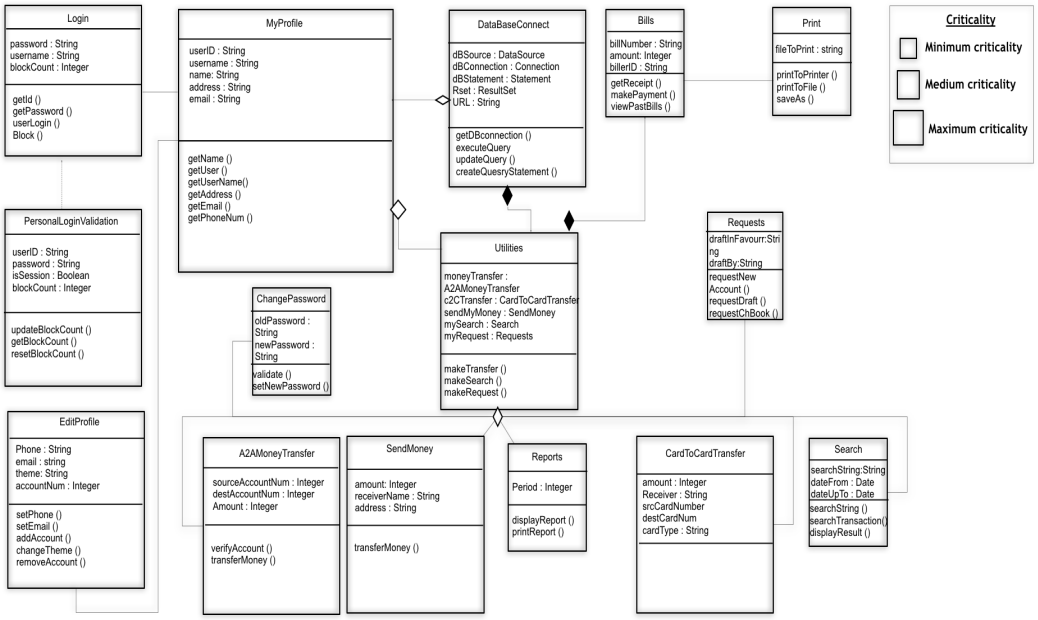

Figure 4.39: A size variation on the IBS class diagram.

Size variation and text labels, separation lines and borders

We have seen that the size variation of an UML node means the variation of its area. But, as glimpsed by **S1** - *Size and brightness*, the size variation is less significant with empty and white spots. In that context, increasing *only* the area of UML nodes, as shown in the figures 4.38 and 4.39, makes them relatively empty. The size of the text is small compared to the area of the concerned UML node. Therefore, to avoid the application of the size to a relatively empty UML nodes, three possibilities of implementations are available.
The first possibility consists of varying not only the area but also the size of the border (i.e., line thickness) with or without varying the thickness of the separation lines, as depicted in Figure 4.40.

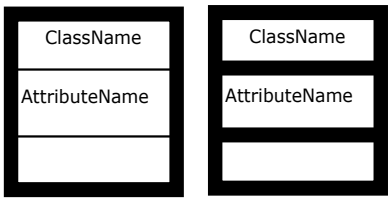

Figure 4.40: A size variation of the borders and/or separation lines on an UML class.

However, such variation can be perceived as a non-significant size variation between the text in one hand and the border and/or the separation lines in the other hand. As the size retinal variable is dissociative (**S3** - *Size and dissociation*), such variation dissociates the text from the borders and separation lines, which prohibits the visualization of an UML node as one entity. It is likewise for the second implementation where we vary only the size of the contained text, as shown in Figure 4.41.

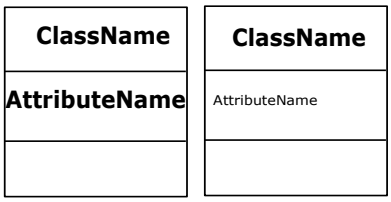

Figure 4.41: A size variation of only the text of an UML class.

As a result, to avoid applying the size variation to a relatively empty UML nodes (**S1** - *Size and brightness*) and to take into account the dissiociative perceptive attitude of the size retinal variable (**S3** - *Size and dissociation*), we generate the following guideline:

> **[SoG-UML #23] The size retinal variable should be applied not only to the area of UML nodes, but also to the text, separation lines and borders.**

The effective implementation is then depicted in Figure 4.42 on an UML class. Figure 4.43 shows it in the use cases, decisions and activities. Figure 4.44 shows the effective implementation of the size retinal variable in the IBS class diagram where the size variation is employed to express the levels of critically of each class.

Figure 4.42: The effective implementation of the size retinal variable to a UML node.

Figure 4.43: Effective implementations of the size retinal variable to four UML nodes.

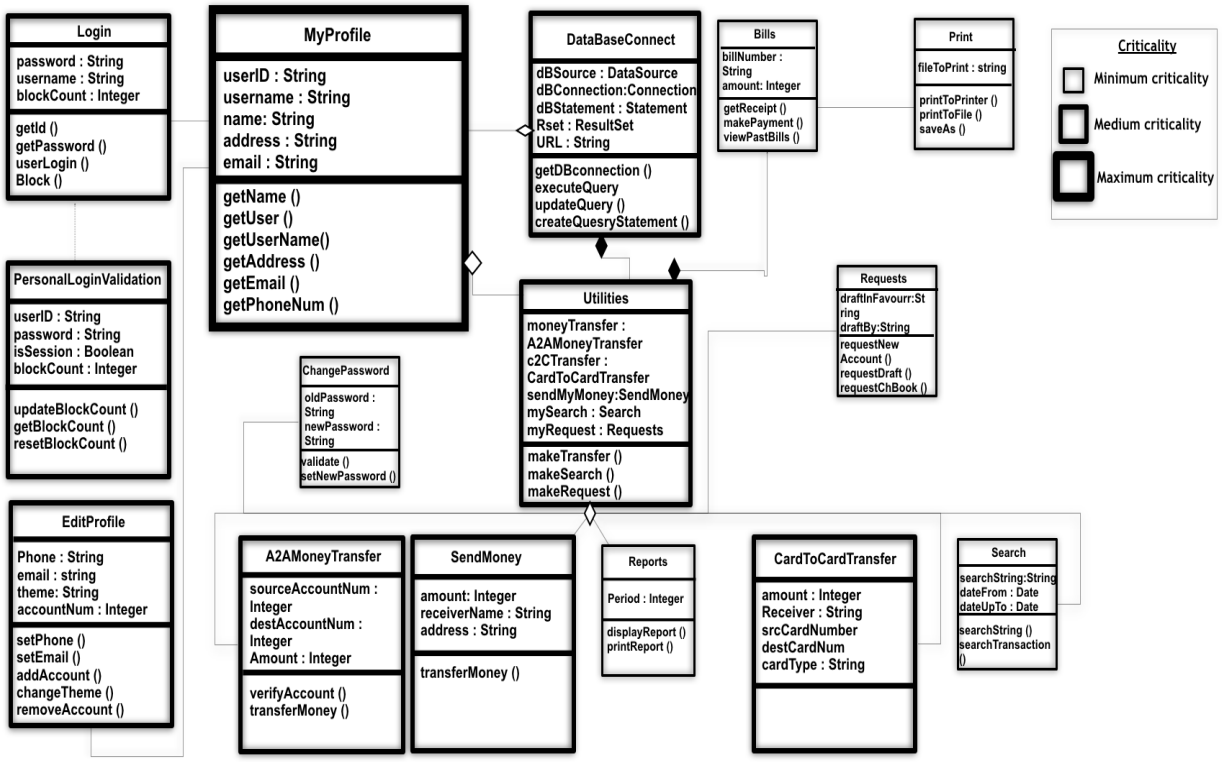

Figure 4.44: The effective implementation of the size retinal variable in the IBS class diagram.

4.5.2 B- UML diagrams that do no not support a size variation

Size variation on the main UML node and text, separation lines and borders

If the UML diagram has a relatively big visual density. Varying the area of the UML nodes might lead to the impossibility of its visualization in one single view. UML nodes are then assimilated to zones of the SoG. In that context, as stated by **S1** - *Size and brightness*, the size variation consists on changing the size of its constituents, points or lines can vary in number or size.

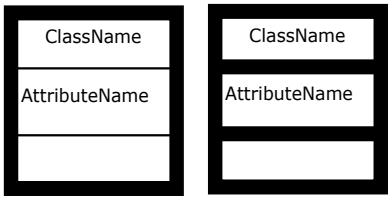

Figure 4.45: varying the size of only the separation lines and borders.

For that, varying the size of the UML node's constituents can be executed in four different manners, as illustrated in the Figures 4.45, 4.46 and 4.47. The first way is depicted in Figure 4.45. We vary the size of only the border without and with lines of separation (i.e., without varying it area). But, such variation might hamper the visualization of an UML node as one entity, because of the dissociative property of the size variation (**S3** - *Size and dissociation*).

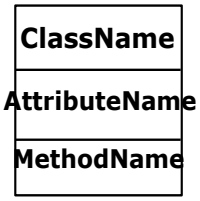

Figure 4.46: Varying the only the size of the text of an UML class.

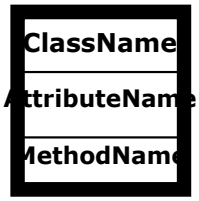

Figure 4.47: Varying the size of the border and the text.

Another possibility of implementation is depicted in Figure 4.46. Only the size of the contained text of an UML node is increased (i.e., without changing the area). In such case, the text might be very long, it will exceed the border and turning back to the next line will need to increase the height of the UML node. The third way is illustrated in Figure 4.47. The size of the border and the text are increased in a constant area. This implementation might hinder the readability of the text when categories of big sizes are used.

In that context, as explained by **S4** - *Size and combinations of forms*, using different forms of spots (i.e., points, lines and zones) guarantees a high selectivity. It is easy to use and very effective. In UML, in order to benefit from the selectivity of different forms of spots, we propose to add a point in the right corner of the graphic element. Hence, changing the size of the point may be sufficient to effectively perceive and understand the size variation as illustrated in Figure 4.48.

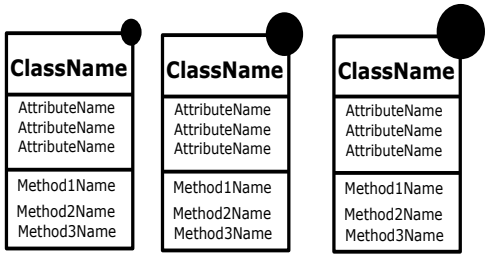

Figure 4.48: Effective implementation of the size retinal variable, in case the area of the corresponding UML node cannot vary.

Figure 4.49 illustrate the effective implementation in the IBS class diagram.

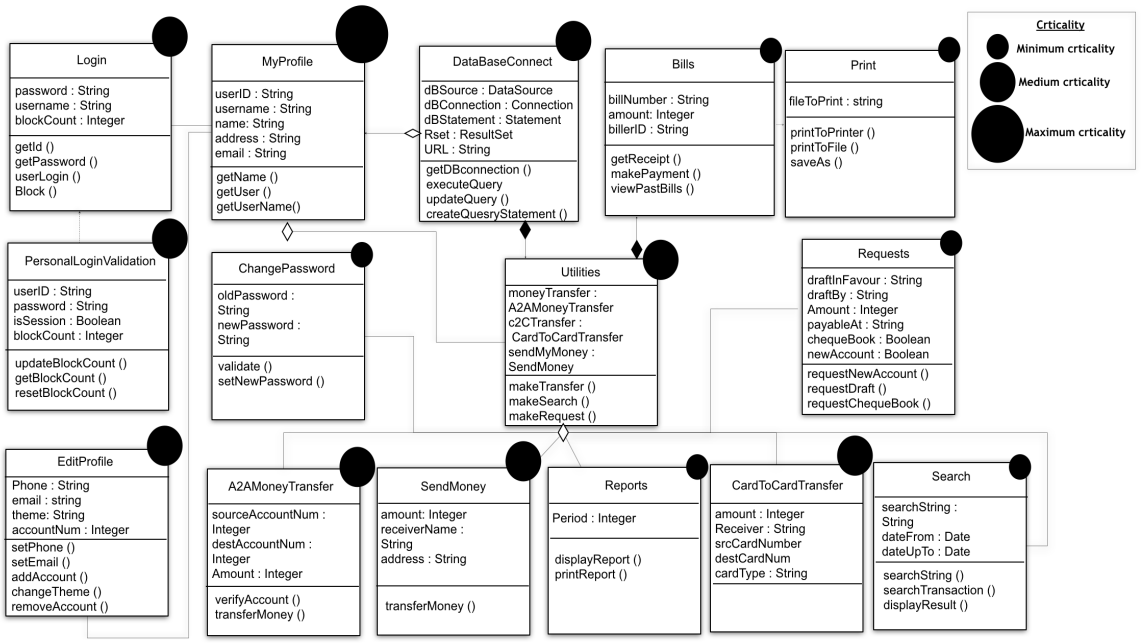

Figure 4.49: Effective implementation of the size retinal variable on the IBS class diagram.

We prescribe then the following guideline:

> **[SoG-UML #24] The size retinal variable should be applied to an additional point in the right corner of the UML node.**

4.5.3 Size variation and icons

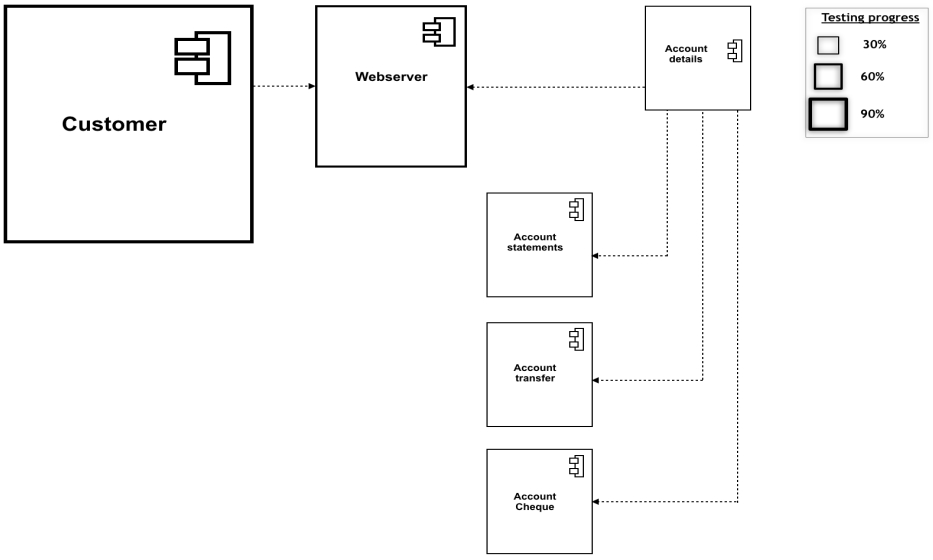

Figure 4.50: Effective implementation of the size retinal variables on icons.

To avoid applying the size retinal variable to relatively empty UML nodes (**S1** - *Size and brightness*), we propose to increase also the size of the contained icons. Oth-

erwise, they will induce to a dissociative perception of the corresponding UML node, as pointed by **S3** - *Size and dissociation*. For that, both the borders and the area of icons should vary their size, as shown in Figure 4.50.

We conclude then the following guideline:

> **[SoG-UML #25] To ensure a better selectivity, the icon's area should vary proportionally with the area of its related UML node.**

4.5.4 Size variation on satellites/ports

Varying the area of a port is obvious when applying the size retinal variable to it. The latter variation refers to the definition of the size retinal variable, as mentioned by **B6** - *Definition*. We chose to vary their line thickness to reinforce their perception against any possible background. For that, we define the following guideline:

> **[SoG-UML #26] Varying the size of a port implies varying its area and its border lines thickness.**

4.6 SoG-UML: Grain

Table 4.4 explains the used rules to justify our guidelines about the effective use of the grain retinal variable in UML.

Table 4.4: The used rules from the SoG to generate our guidelines about the grain in UML.

Rule identifier	Title	Rule	Explanation
B4	Definition	**The grain corresponds to the magnification factor applied on a particular texture.**	It mainly consists of a *zoom* on a particular texture. [**SoG-79**]: *Succession of photo-graphical reductions of a semis of spots. In a specific surface and for a regular semis, these reductions increase the number of spots, without varying the brightness.* ***In a particular brightness****, the grain is the quantity of separable spots contained in a unit surface.*
B2	One-to-one mapping of information properties	**In a diagram, grain should express one single information property.**	During the analysis process, each information property should be mapped to one single retinal variable [3] which has (at least) the same organization level. [**SoG-10**] *It is important to transcribe each information property by one visual variable which has at least the corresponding organization level and the length.*
G1	Grain and size	**The bigger the spot is, the bigger the number of selective categories of grain is.**	As for any retinal variable, the bigger its size is, the better its visual variation is perceived. [**SoG-73**]: *In points, the grain needs big spots and do provide only two or three selective categories.*

Table 4.4 continued from previous page

G2	Null grain	**The null grain might disappear from any graphical representation and the information it represents will also disappear.**	The null grain is composed of multiple small spots which are invisible to the naked eye. It is difficult to reproduce it after printing for example. [**SoG-73**]: *The null grain cannot be reduced and disappears from any microfilmed reproduction.*

4.6.1 Grain variation on the Main UML node

Figure 4.51 illustrates the possible implementations of the grain retinal variable to an UML node (for the texture composed of small chronometers).

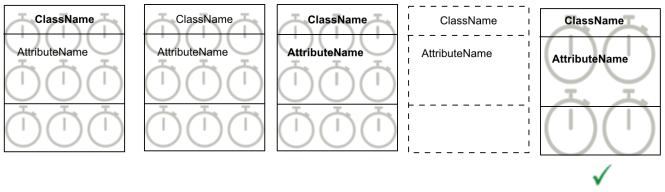

Figure 4.51: Possible implementations of the grain visual variable to an UML class.

Based on **B4** - *Definition*, the grain is assimilated to a zoom that we apply to a particular texture. The grain variation depends on the area of the spot in question. The bigger it is, the higher the number of selective categories of grain is (**G1** - *Grain and size*). In that context, the grain variation might be applied to the borders and separation line of UML nodes, as depicted in Figure 4.52. However, they are relatively thin which decreases the number of selective categories of grain. In addition, this variation is already locked by the UML primary notation, as illustrated in Figure 4.1.

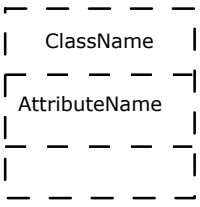

Figure 4.52: Applying the grain to the border lines of a UML class will alter the UML primary notation.

Then, the only other possible implementation consists of varying the grain of the UML node's background, which represents the biggest area. Such implementation allows to build the biggest number of selective categories of grain (**G1** - *Grain and size*). We generate the the following guideline:

> **[SoG-UML #27] The grain retinal variable should be applied to the background of an UML node.**

In that context, many textures are available, for which we can apply a grain variation. Three examples of textures are depicted in Figure 4.53 below.

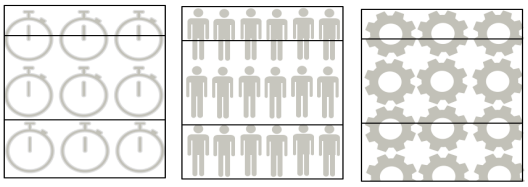

Figure 4.53: Three examples of possible textures.

4.6.2 Grain variation and text, separation lines and borders

The grain variation is problematic in UML because it might interfere with the readability of the contained text. But also, the spots of the used texture should be easily identifiable to the naked eye. They might be significant to the reader (e.g., a chronometer to express time). A designer should then avoid using null grain.

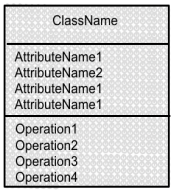

Figure 4.54: Null grain and the readability of text.

Null grain denotes a category of a grain variation where a lot of very small spots, invisible to the naked eye, are used as shown in Figure 4.54. Besides, this grain is not recommended by the SoG because it is difficult to build and cannot be reproduced after printing (**G2** - *Null grain*). We generate then the following guideline:

> **[SoG-UML #28] Null values of grain where a lot of small spots are used should be avoided in UML.**

Based on **B4** - *Definition*, the grain variation is created on a particular level of brightness. This parameter can then be controlled to better ensure the readability of the text of UML nodes. In that context, high levels of brightness are the best levels which guarantee the visibility of the the black text, because of the big contrast between both of them (i.e., black and white.

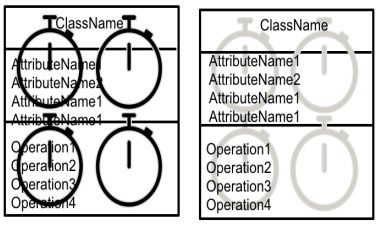

Figure 4.55: Grain and brightness.

Figure 4.55 shows how the brightness of the used texture affects the readability of the text in an UML class (i.e., the class in the left has a low level of brightness and the class in the right has a high level of brightness). We describe then the following guideline:

> **[SoG-UML #29] High levels of brightness should be applied to the used textures when applying grain variation.**

Figure 4.56 illustrates the effective implementation of the grain retinal variable on the IBS class diagram. The grain is used to express the ordered information property *progress of the implementation*. As shown in the associated keys, three categories of grain are employed.

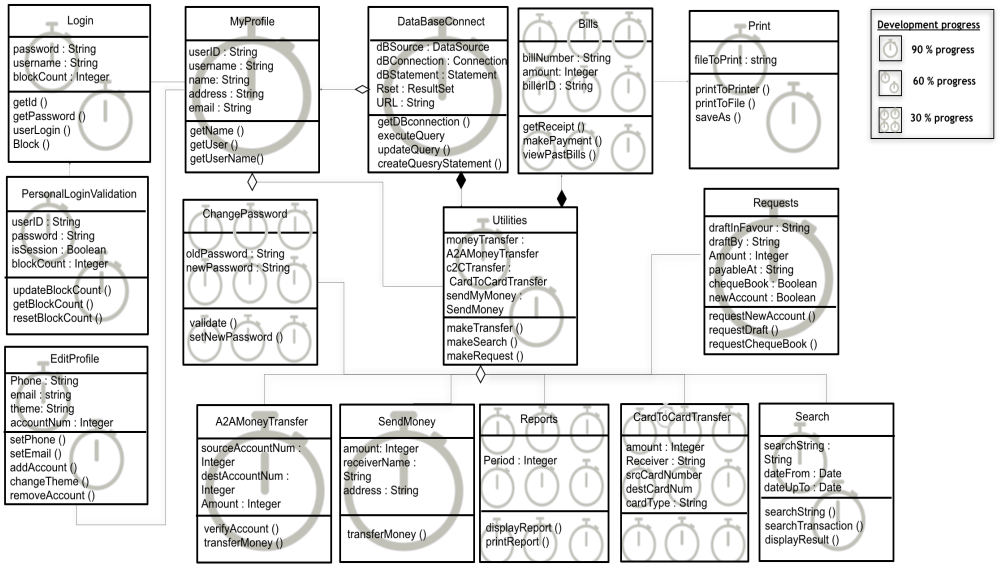

Figure 4.56: The effective implementation of the grain retinal variable on the IBS class diagrams.

Figure 4.57 shows the chosen implementation of the grain retinal variable in different UML nodes.

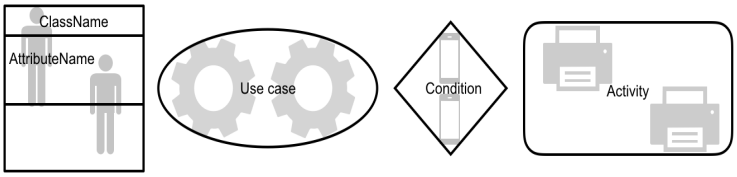

Figure 4.57: Effective implementations on other UML nodes.

4.6.3 Grain variation and icons

Based on **G1** - *Grain and size*, the smaller the spot is, the less the number of selective categories of grain is. In that context, the area of UML icons is relatively small to effectively support a grain variation. A zoom on a particular texture on them will not be accurately perceived. We conclude then the following guideline:

> **[SoG-UML #30] The background of icons should not have a grain variation.**

Figure 4.58 illustrates the effective implementation of icons in the IBS component diagram.

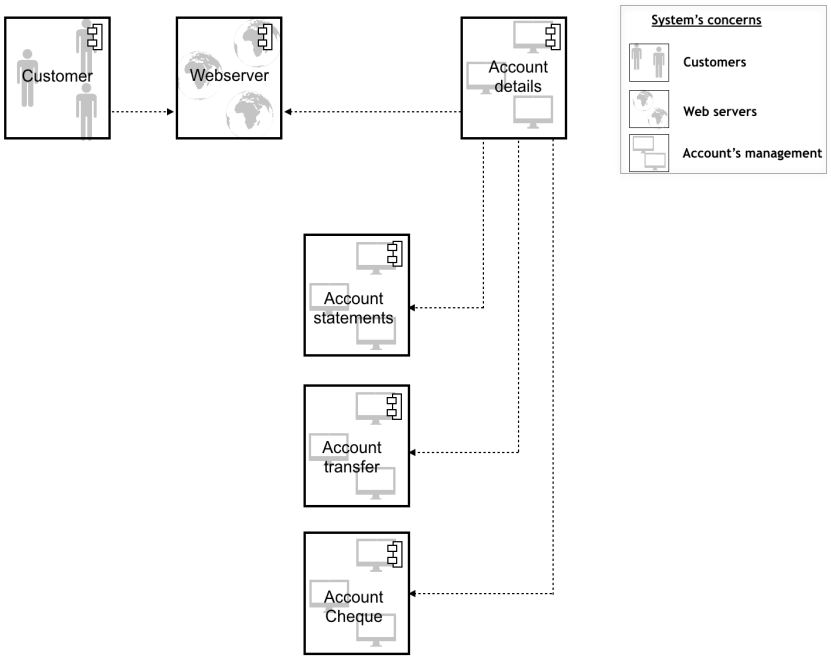

Figure 4.58: The effective implementation of the grain (here different textures) retinal variable and icons.

4.6.4 Grain variation on satellites/ports

As illustrated in figure 4.2, the grain variation of the satellites is locked by the UML primary notation (i.e., ports having a cross or not). Therefore, satellites have not to vary their grain, as stated by our next guideline:

> **[SoG-UML #31] Satellites have not to change their grain. This variation is locked by the UML primary notation.**

4.7 SoG-UML: Orientation

Table 4.5 includes all the rules of the OsG that we have used to find the guidelines of effectiveness regarding the orientation retinal variable in UML.

Table 4.5: The used rules from the SoG to generate our guidelines about the orientation variation in UML.

Rule identifier	Title	Rule	Explanation
B5	Definition	**The difference of angles between multiple parallel spots constitutes the stimuli of the orientation variation.**	An example of a variation of orientation is the transition between a line and its perpendicular. There is a variation of orientation of 90 degrees between both of them.[**SoG-93**]: *A spot (i.e., a point, line or zone) can take an infinity of different orientations without changing its center.*

Table 4.5 continued from previous page

O1	Orientation and forms of the spot	**The human eye can feel the differences of orientations as far as the point represents a linear aspect.**	Non linear spots would not stimulate the reader to an orientation variation. [**SoG-93**] *We can feel the differences of orientations as far as the point represents a linear aspect (The ratio Height/basis has to be at least 4/1)*
O2	Orientation and zones	**Varying the orientation in big areas (zones) is the less selective variation.**	In zones, the variation of orientation consists at building many parallel spots (e.g., lines) in the whole area of a zone. [**SoG-93**] *In zones, the orientation variation is the easiest variation to construct, but at the same time, it is the less selective variation. It has to be combined with another selective variable to ensure the rapid perception of its variation*

4.7.1 Orientation variation on the main UML node

Based on **B5** - *Definition*, it is the difference of angles between multiple parallel spots which constitutes the stimuli of the orientation variation. In that context, as stated by **01** - *Orientation and forms of the spot*, the human eye can feel the differences of orientations as far as the point represents a linear aspect. To find the effective implementations regarding these rules, we begin by varying the orientation of the text contained in UML nodes as illustrated in Figure 4.59.

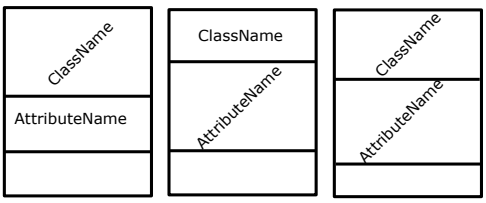

Figure 4.59: Varying the orientation of the contained text of an UML node.

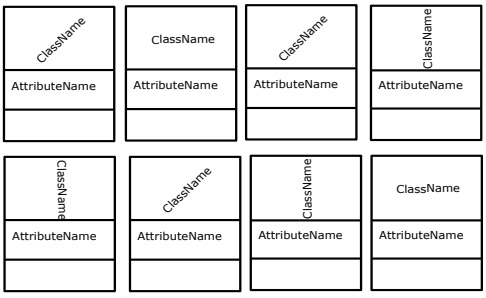

Figure 4.60: No phenomena of multiple parallel spots.

However, such implementation would not create the phenomena of multiple parallel spots **B5** - *Definition*. In fact, in the same UML node, we will find two angles of orientations: the significant variation of orientation (of the name or the rest of the text) and the horizontal (default) angle. This fact might disturb the selective perception, as shown in Figure 4.60. In this case (Figure 4.60), all the attributes compartment have

a linear aspect (rectangle). However, in practice, they might have different shapes (rectangles or squares) depending on the contained text of each compartment. The same observation is generated by varying the orientation of the whole contained text of an UML node and the separation lines, as illustrated in Figures 4.61 and 4.62.

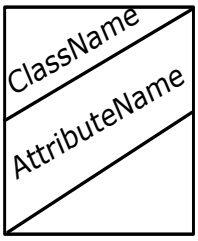

Figure 4.61: Varying the orientation of the text and the lines of separation.

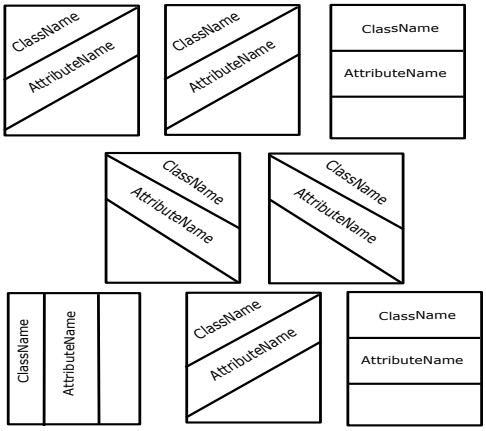

Figure 4.62: No phenomena of multiple linear spots.

In Figure 4.63 below, the orientation of the whole UML node is varied. In this case, the UML node does not have a linear aspect (i.e., the ratio Height/basis is less than 4/1) (**01** - *Orientation and forms of the spot*). Then, the orientation variation will not be rapidly perceived.

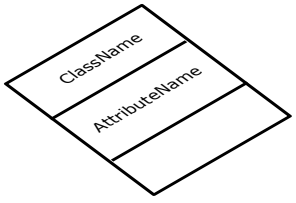

Figure 4.63: Changing the orientation of the whole UML node is problematic for UML nodes having non-linear aspect.

In this case, we propose the following guideline:

> **[SoG-UML #32] If the UML node has a linear aspect, the orientation of the whole UML node can be changed.**

Figure 4.64 illustrates the effective implementation of UML nodes having a linear aspect.

Figure 4.64: Varying the orientation of UML nodes having a linear aspect.

Now, if the UML node does not have a linear aspect, the orientation might be applied to the background of UML nodes, as depicted in Figure 4.65.

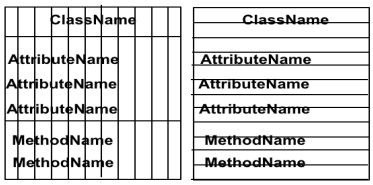

Figure 4.65: Varying the orientation of a constant texture.

In such case, as stated by **O2** - *Orientation and zones*, the orientation variation the variation which is the less selective among the other retinal variables. In addition, such variation will significantly impact the readability of the text (Figure 4.65). We conclude then the following guideline:

> **[SoG-UML #33] The orientation retinal variable should not be applied to the background of UML nodes.**

4.7.2 Orientation variation and text, separation lines and borders

As illustrated earlier, the text labels, separation lines and borders vary their orientation in UML nodes having a linear aspects. The following guideline is then precised:

> **[SoG-UML #34] Text, separation lines and borders should all vary their orientation for UML nodes having linear aspects.**

4.7.3 Orientation variation and icons

The orientation of icons is locked by the UML primary notation. If the main UML node changes its orientation, the icon should take the same angle of variation, as stated by the following guideline:

> **[SoG-UML #35] The orientation of icons should always have the same orientation of the main UML node.**

4.7.4 Orientation variation and satellites

The orientation of the satellites should be the same as the UML node in which it is attached. In fact, their orientation might change their shape. For example, a square might be accidentally transformed to a diamond by a 90 degrees orientation.

[SoG-UML #36] The orientation of the satellites should be the same as the UML node in which it is attached.

4.8 Superposition of UML nodes

We have studied the use of the retinal variables on UML nodes which are contained in a white background. The present section treats the containment of UML nodes in other ones (e.g., a class in a package). There exist 25 possibilities of containment between UML nodes having a retinal variable variation. In fact, we have five retinal variables and for each one, we study its variation on all the possible variations of the background. Therefore, for each retinal variable, five possibilities of containment have to be carefully discussed. The first column of each table is in the form of Content/Container. *Content* refers to the UML nodes which are contained in a container. *Container* represents any UML node which might contain other ones (e.g., package, fragment). We chose to illustrate these variations via tables to see how we treat both of the container and the content for each possibility of superposition. They are put in parallel column in each Table. This presentation allows also to easily navigate between the different tables.

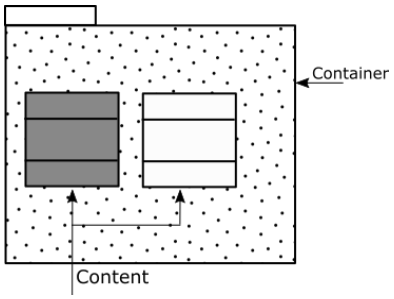

Figure 4.66: Example of a UML classes with a brightness variation and contained in grained package.

Figure 4.66 shows an example of brightness variation of the classes (i.e., contents) on a grained background (i.e., container). The latter variation is denoted Brightness/Grain. Table 4.6 below details the rules of the SoG that we used to generate our guidelines.

Table 4.6: The used rules from the SoG to generate our guidelines about the superposition of the retinal variables in UML.

Rule identifier	Title	Rule	Explanation
Super-1	Relationship between the background and the main information	**The background of a representation should be brighter than the spots that represent a particular information.**	UML diagrams are assimilated to maps of the SoG: The container in the background and the contained UML nodes are superimposed to it. [**SoG-327**]: *A Linear selective problem in maps leads then, from the beginning to two categories of visibility, the one, the brightest possible is dedicated to the background, the other one, the stronger, is dedicated to the original information.*

Table 4.6 continued from previous page

Super-2	Superposition of colors	**Two superimposed spots should have different colors from each other.**	Colors are perceived thanks to the difference between two areas having different colors. [**SoG-89**]: *To get a color sensation, which means a significant color difference, we must have a covered surface with a uniform color (the same in the whole surface) and which has the opposite color (in the spectrum) to another surface, with another uniform color.*
Super-3	Superposition and brightness	**The background which contains a brightness variation should be white.**	The length of the brightness depends on the available distance between the black and white. [**SoG-73**]: *This latter will be decreased if the white is not white. It is therefore a major mistake to color in gray, blue, green, red, etc., the paper of a graph or a map in which the brightness variation is significant*

4.8.1 Color: containment of UML nodes having a color variation on all the possible containers

Table 4.7 discusses the containment of colored UML nodes in all possible backgrounds.

Table 4.7: Containment relationship between UML nodes and colors.

Content/container	Content	Container
Color/Color	[**SoG-UML #37**] **To get the color sensation, choose colors that are different from the container's color**. As for the container, UML nodes should not have the same color as their container (**Super-2** - *Superposition of colors*).	[**SoG-UML #38**] **To ensure a better visual selectivity, the container should have a different color from the colors of its contained elements.** Based on **Super-2** - *Superposition of colors*, colors are perceived thanks to the difference between two areas having different colors. The container should then have a different color from its contained UML nodes. Such variation induces to a color sensation to the reader. That allows us to state that the container and its contained UML nodes should have different colors from each other. Levels of brightness of each one are discussed in Table 4.9.

Content/container	Content	Container
Color/Brightness	**[SoG-UML #39] Choose colors having all the same brightness which is opposable to the brightness of the background.** If the container has a big value of brightness, the contained elements should have colors having small value of brightness and vice versa (**Super-1** - *Relationship between the background and the main information*).	**[SoG-UML #40] The container should be as brighter as its contained elements are dark and vice versa.** The brightness variation might correspond to a series of grays (i.e., from white to black) or to a particular hue (i.e., from light to dark), as explained by **B3** - *Definition.* What matters is that the contained elements might be rapidly visually selected. For that, the container should be as brighter as their contained elements are dark and vice versa (**Super-1** - *Relationship between the background and the main information*).
Color/Grain	**[SoG-UML #41] Increase the thickness of the borders and the lines of separation to better select the contained UML nodes in a possible very low value of grain.** When the background of the container has a null grain (**G2** - *Null grain*), the borders of the contained UML nodes will be no more easily and rapidly selected. That might alter the primary notation: Viewers will not be able to relatively rapidly select the exact shape of the concerned contained node. Consequently, we propose to relatively increase the thickness of the border lines, separation lines text of the contained UML nodes to better select them from the background (**Super-1** - *Relationship between the background and the main information*). The line thickness of all the contained elements should be the same (no size variation).	**[SoG-UML #42] Choose big values of grain for containers having more contained elements.** In fact, the container is assimilated to the background of a map. It has to ensure the selectivity of its contained elements (**Super-1** - *Relationship between the background and the main information*).
Color/Size	**No particular rules.** The fill of the background does not change.	**No particular rules.** The size variation of the container does not harm the selectivity (visibility) of the colored contained elements as long as the variation does not affect their size (See Table 4.8 size/size: the more the spot is small, the less the colors are visually separable).
Color/Orientation	**No particular rules.** The fill of the background does not change.	**No particular rules.** The fill of the background does not change.

4.8.2 Size: containment of UML nodes having a size variation on all the possible containers

Table 4.8 discusses the containment of UML nodes presenting a size variation in all possible backgrounds.

Table 4.8: Containment relationship between UML nodes and sizes.

Content/container	Content	Container
Size/Size	**[SoG-UML #43] The size of the contained elements must be proportional to the size of their container.** As explained by **S1** - *Size and brightness*, the size variation loses all its perceptive properties with empty spots. Very small sizes of the contained elements (compared to their "very big" container) make the container relatively empty. **[SoG-UML #44] The biggest area of the contained elements is controlled by the size of its container.** These latter must not exceed the borders of the corresponding container. **[SoG-UML #45] There is a limit under which the UML nodes are no more visible.** When the size of the container decreases, the sizes of the contained elements might decease. But, there is a limit under which the contained UML nodes are no more visible. Their contained text have to be always easily readable. In addition, the layout of the contained elements and their related edges have to still be bright and readable (no intersections, texts related to the edges are readable, text inside the contained UML nodes is readable).	**[SoG-UML #46] The smallest size of the container is controlled by its contained UML nodes.** In fact, there is always a limit under which the contained elements might not be sufficiently bright and the layout might be altered.
Size/Color	**No particular rules.**	**No particular rules.** The color variation of the container does not impact the size variation of its contained elements.
Size/Brightness	**No particular rules.**	**No particular rules.** The brightness variation of the container does not impact the size variation of its contained elements.

Content/container	Content	Container
Size/Grain	**No particular rules.**	**No particular rules.** The grain variation of the container does not impact the size variation of its contained elements.
Size/Orientation	**No particular rules.**	**No particular rules.**

4.8.3 Brightness: containment of UML nodes having a brightness variation on all the possible containers

Table 4.9 presents the possibilities of containment between UML nodes presenting a brightness variation in a all possible containers variations.

Table 4.9: Containment relationship between UML nodes and brightness.

Content/container	Content	Container
Brightness/Color	[**SoG-UML #47**] **Colors of the contained UML nodes should have a smaller level of brightness compared to their container.** The contained elements have all the same value of brightness. The latter value of brightness should be smaller than the one used for their container. This is to ensure better selectivity.	[**SoG-UML #48**] **The container of UML nodes which have a significant brightness variation should be white.** On one hand, as defined in **B3** - *Definition*, the brightness variation might range from black to white via colors. On the other hand, selective colors should have the same value of brightness (**C1** - *Effective categories of colors*). In that context, the SoG proves that the length of the brightness depends on the available distance between the black and white. Based on **Super-3** - *Superposition and brightness*, it is a major mistake if graphic authors color in gray, blue, green, red, etc., the paper of a graph or a map in which the brightness variation is significant. The SoG says that it is better to apply the brightness in a white background.
Brightness/ Brightness	[**SoG-UML #49**] **The contained UML nodes should have a smaller value of brightness than their container.**	[**SoG-UML #50**] **The container should be brighter than its contained UML nodes.** From the SoG point of view, the container should be white **Super-3** - *Superposition and brightness*. We propose that the container have to be brighter than its contained elements.

Content/container	Content	Container
Brightness/Grain	**No particular rules.**	[**SoG-UML #51**] **The container should not have a grain variation or a very big value of grain where a big zoom is applied to the spots of the used texture.** The grain variation of the container might be hidden by its contained elements. In addition, as recommended by **Super-3** - *Superposition and brightness*, the background of a brightness variation should be white.
Brightness/size	**No particular rules.** The size variation of the container does not affect the brightness of its contained UML nodes.	**No particular rules.**
Brightness/ orientation	**No particular rules.**	**No particular rules.**

4.8.4 Grain: containment of UML nodes having a brightness variation in all the possible containers

Table 4.10 discusses all the possibilities of containment between UML nodes presenting a grain variation and all possibilities of containers variations.

Table 4.10: Containment relationship between UML nodes and grain.

Content/container	Content	Container
Grain/Grain	**No particular rules.**	[**SoG-UML #52**] **The container should have a maximum value of grain.** Big values of grain means that many big spots are scattered in the background of the container (**G2** - *Null grain*). It guarantees many white spaces. Such variation ensures better readability and selectivity of the contained UML nodes.
Grain / Brightness	[**SoG-UML #53**] **If the container is dark, then the grain variation has to be as brighter as possible.** Such perception could be achieved by a redundant combination **Combi-1** - *Redundant combination* with the brightness - if possible - or with color.	[**SoG-UML #54**] **The container should have a brightness as brighter as possible.** The container have to be as brighter as possible to allow the reader's eye to select the contained UML nodes (**Super-3** - *Superposition and brightness*).

Content/container	Content	Container
Grain/Size	**No particular rules.** The size variation does not affect the background of the container, so the effective implementation of the grain in white background is applied.	[**SoG-UML #55**] **The size variation should not affect the grain variation of the contained nodes.** As defined, the grain is succession of photo-graphical reductions of a semis of spots. It might be assimilated to the zoom variation (**B4** - *Definition*). If the size variation of the container affects the size of its contained elements, the latter variation should not alter the grain variation. This is by adding the same degree of zoom or de-zoom to all the contained elements to ensure the same distance between the categories of grain.
Grain/Color	**No particular rules.**	**No particular rules.**
Grain/Orientation	**No particular rules.**	**No particular rules.**

4.8.5 Orientation: containment of UML nodes having an orientation variation on all the possible containers

We observe that no particular rules for the orientation variation in the different possible backgrounds. The effective implementations for a white background are applicable in all cases.

4.9 Combinations of the retinal variables

Table 4.11 includes the rules of the SoG that we have used to discuss the effective use of the combinations of the retinal variables in UML.

Table 4.11: The used rules from the SoG to generate our guidelines about the combinations of the retinal variables in UML.

Rule identifier	Title	Rule	Explanation
Combi-1	Redundant combination	**Redundant combinations are used to express one single information property.**	All the retinal variables of a combination express one single information property. [**SoG-187**]: *A combination of multiple variables, used to transcribe only one information property, is a redundant combination.*
Combi-2	Significant combination	**Significant combinations are used to express different information properties.**	Each retinal variable expresses one single information property. [**SoG-189**]: *When in the same spot, two variables are used and each one is assigned to express a different information property, the combination is called "significant".*

Table 4.11 continued from previous page

Combi-3	Perceptive properties of a combination	**A combination of retinal variables holds the perceptive properties of the highest one in the Table 3.5**	The retinal variables are ordered as follows: Size and brightness have the highest level but they are both dissociative. Then we find the grain, colors, orientation and finally the shape. [**SoG-186**]: *A combination of variables has the properties of the highest organization level, defined by the Table 3.5. Except the combinations of brightness and size, both of them are dissociative.*
Combi-4	Combination of size and brightness	**A combination of the size and the brightness retinal variables reinforces the readability of quantitative information.**	This combination can be used as long as the information supports a dissociative perception. [**SoG-71**]: *A combination of the size and the brightness retinal variables can be used to reinforce the readability of quantitative information.*

We treated the effective use of the retinal variable in a white background and all the possibilities of superposition between each other. In this Section, we study the effective combinations of the retinal variables in UML nodes. In fact, the SoG identifies 63 possible combinations between these latter. An example of combination between the shape (i.e., a square), the color (i.e., Cyan) and the size retinal variables is shown in Figure 4.67.

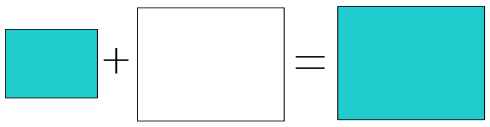

Figure 4.67: An example of a combination between the size and the color retinal variables.

Combinations of the retinal variables might serve to describe one single information property. Such combination is called redundant combination (**Combi-1** - *Redundant combination*). Combinations might also be used to describe different information properties where each retinal variable concerns one single information. It is called a significant combination (**Combi-2** - *Significant combination*). At this level, one question arises: what are the perceptive properties of the combination of different retinal variables ?

A redundant combination holds the organization level and the length of the retinal variable which has the highest organization level, as depicted in table 3.5. For example, a combination of shape and color holds all the perceptive properties of color. In fact, the color is higher than the shape in Table 3.5 (**Combi-3** - *Perceptive properties of a combination*).

4.9.1 Combinations of the retinal variables in UML

Effective implementation of a combination of retinal variables in UML

Figure 4.68 illustrates an example of a redundant combination of the color and the grain retinal variables. It represents the union of the effective implementation of both

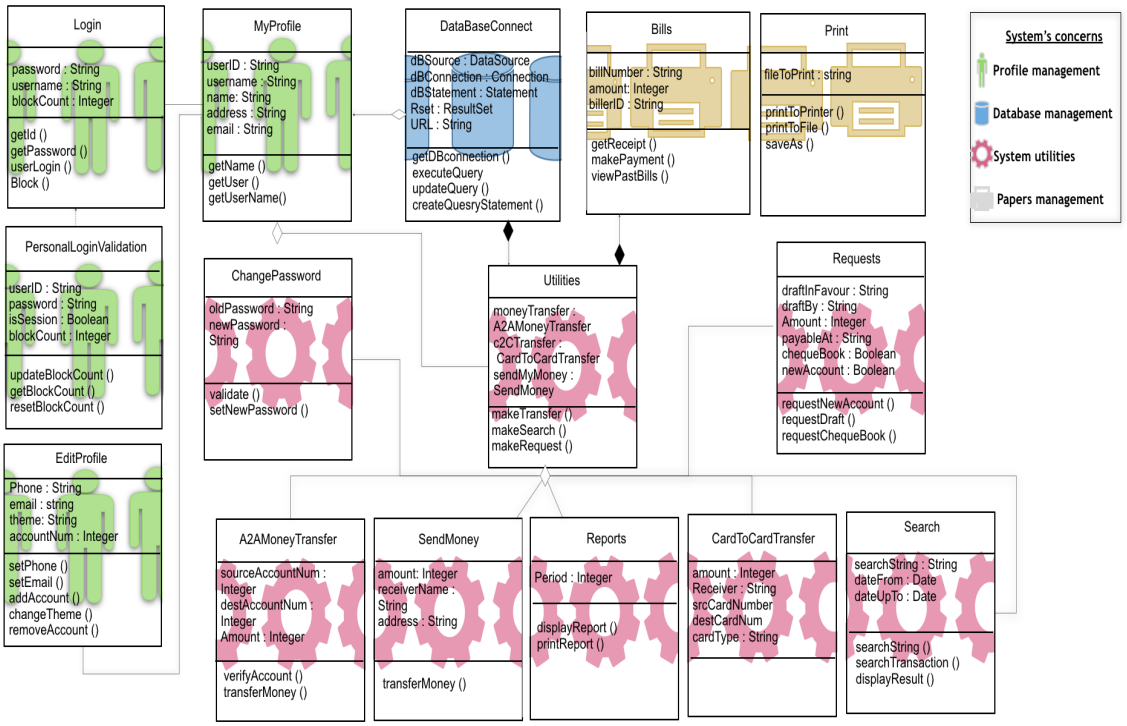

Figure 4.68: Redundant combination of colors and grain in the IBS class diagram.

retinal variables. In one hand, the effective implementation of colors is its application to the background of an UML node. In the other hand, the effective implementation of the grain retinal variable is to apply it to the background of an UML node also. As a result, the effective implementation is to to apply the color to the background grain. the following guideline stems then:

> **[SoG-UML #56] The effective implementation of a combination between the retinal variables should be the union of the effective implementation of each one.**

Combinations of size and brightness

The brightness is ordered, but it is also dissociative. It can be combined with the size to reinforce the perception of quantitative information, as stated by **Combi-4** - *Combination of size and brightness*. The following guidelines are then generated:

> **[SoG-UML #57] The size is the only quantitative retinal variable. It should be used to express quantitative information.**

> **[SoG-UML #58] Combination of size and brightness should be used to reinforce the readability of quantitative variations.**

Figure 4.69 illustrates such combination on the IBS class diagram. A redundant combination between the size and the brightness retinal variables. It serves to express the quantitative information property *progress of the development*. In that context, the size is the most selective retinal variable. It might be combined with color, brightness, grain, orientation or shape to reinforce visual the selection. However, authors of

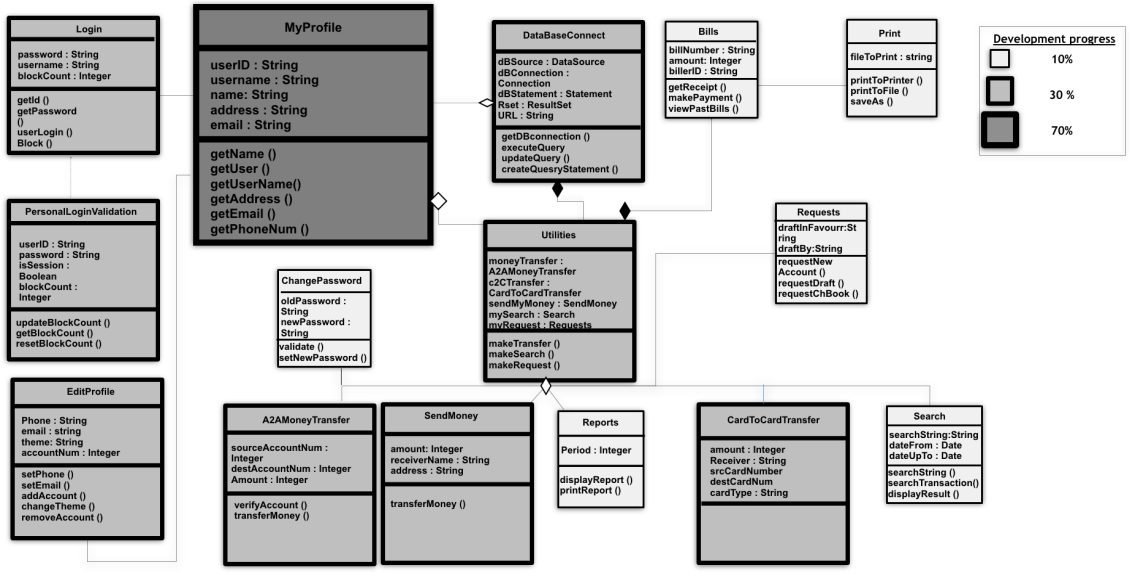

Figure 4.69: Size and brightness to reinforce the quantitative perception in the IBS class diagram.

graphical representations have to be careful to the dissociative perception that it might cause. In fact, because they are dissociative, the size and the brightness can exclude the associative perception. They will dominate all the combinations done with them. We generate then the following guidelines:

> **[SoG-UML #59] Size should not be used when an associative perception is required**

> **[SoG-UML #60] Brightness should not be used when an associative perception is required**

In that context, the grain variation is the only retinal variable which can be used to express visual ordering without changing the visibility of spots (i.e., without visual dissociation), as describes the following guideline:

> **[SoG-UML #61] The grain should be used to show visual ordering if the information doesn't support a dissociative perception.**

4.10 Summary

In this chapter, we defined SoG-UML which is a SoG based framework to visually enrich UML diagrams. It relies on the design and action theory [26] to define 61 explicit guidelines about the effective use of the retinal variables in UML. It meets the need of effectiveness in using these variables, which has been stated by the UML practitioners of our previous empirical study (See Chapter 2). It deals with their non-effective usages that we observed in >3500 UML diagrams (different implementations, no keys, non-effective mapping of information properties). The main objective of SoG-UML is to enhance the cognitive effectiveness [4] of UML diagrams by exploiting the benefits of

[4]The cognitive effectiveness denotes the speed, ease, and accuracy with which a representation can be processed by the human mind.

the retinal variables. It complements the incompleteness that we revealed in the CDs and the PoNs frameworks (see Chapter 3). It answers to our research questions: RQ-1, RQ-2 and RQ-3.

For that, we explore the effective implementations of the retinal variables in the UML graphic nodes. This is achieved by managing their graphic particularities which have been identified via our empirical classification of the UML concrete syntax. An implementation of a retinal variable denotes the manner in which it will be applied to a particular UML node (e.g., to its border, background, both). In SoG-UML, we treat not only UML nodes in a white background but also the combinations of the retinal variables on them and their possible containment relationships. The 61 guidelines are complementary to those explained in the first refinement layer of the SoG to UML: the perceptive properties of the retinal variables (i.e., their organization level and length), the analysis, the mapping of the retinal variable to the information properties and the reading process.

The retinal variables might be used by UML practitioners to express information properties in their diagrams in many situations and mainly for communication purposes within the technical team or with their clients. They might use them for their own understanding, to save information about their maintenance tasks or to generate their documentation. Examples of information properties have been mentioned by our practitioners: *ownership of classes, project progression, concerns of the system.*

4.11 Discussion

On one hand, most of the choices of effective implementations seem to be logic and already known. For example, the application of colors to the background of an UML node is very common and represents the existing default configuration in modeling tools. However, our study of >3500 UML diagrams showed that there exist different implementations which are used in practice (e.g., on borders, text name compartment). In addition, there is no scientific framework which formalizes explicit guidelines about such usages. On the other hand, some discussed implementations are not available in the existing modeling tools. For example, changing the orientation of an UML node or applying a grain variation. In our research, to find the most effective implementation of a retinal variable, we chose to be pragmatic. We expose the possible implementations and we justify the most effective ones based on the SoG. In addition, the 61 guidelines should be automated and integrated in the modeling tools. In that way, UML modelers can rapidly and effectively use the retinal variables in their practice. Moreover, even though SoG-UML defines the effective implementations for redundant combinations and superposition of the retinal variables, it is mandatory to precise that such types of utilization are double edged. Their use should not overload the human mind. In that context, an UML diagram has not to present more than two information properties[5]. Finally, as mentioned earlier, SoG-UML relies on the design and action theory[6]. This type of theory requires the definition of methods to validate the guidelines of the proposed theory [26]. For that, we provide SoG-UML as a testable theory and we define then, in the next chapter, a validated design methodology of an experiment to

[5]Layering can be used to display such number of visual variation as Chapter 6 explains later.

[6]A design and action theory is meant to give explicit prescriptions for constructing an artifact (e.g., methods, techniques, principles).

validate the SoG-UML guidelines. We also present prototypes of tools which realize some of the SoG-UML guidelines in the Papyrus environment [24], as a proof of concept ones. They are described in Chapter 6.

Chapter 5

Evaluation

Contents

The validation of SoG-UML principles requires a lot of empirical studies. Figure 5.1 shows the elements that should be taken into account in each validation process. It is displayed in the next page for space contraints.

First, we have five retinal variables with their corresponding perceptive properties. For each retinal variable, the effective implementation on each graphic element, including the main UML node, icons, ports, text, borders and separation lines, the containment relationships and the combinations of the retinal variables should be assessed. In addition, all the UML diagrams should be taken into account. For each type of UML diagram, the visual density variation should be assessed (i.e., small, medium and large). In that context, to succeed the experiment, the scope should be delimited to a few independent variables. For instance, one single retinal variable, one single type of UML diagram, one single graphic element, two implementations on this latter, one single perceptive property and the three possibilities of visual densities might be assessed in one experiment. In such conditions, if we consider the five retinal variables and only the four most used UML diagrams in practice, at least 20 experiments should be conducted.

We define in this chapter a validated design methodology of an empirical study to validate the SoG-UML guidelines. But first, we focus on the lessons learned from an experiment which failed to give statistically significant results. Our validated study consists of a quantitative methodology using an experiment as strategy of inquiry. We also describe the results of the experiment which succeeded to give statistically significant results. It aims at finding the effective implementation of the color retinal variable on UML classes. It compares, on one hand, the implementation of the color retinal variable on the border and, in the other hand, the background of UML classes and the use of text via stereotypes. It empirically validates the effective implementations chosen by SoG-UML, where the color should be applied to the background of an UML node or to a relatively thick border. It also shows that the use of colors helps finding the correct answers in a relatively short time compared to the use of text via stereotypes.

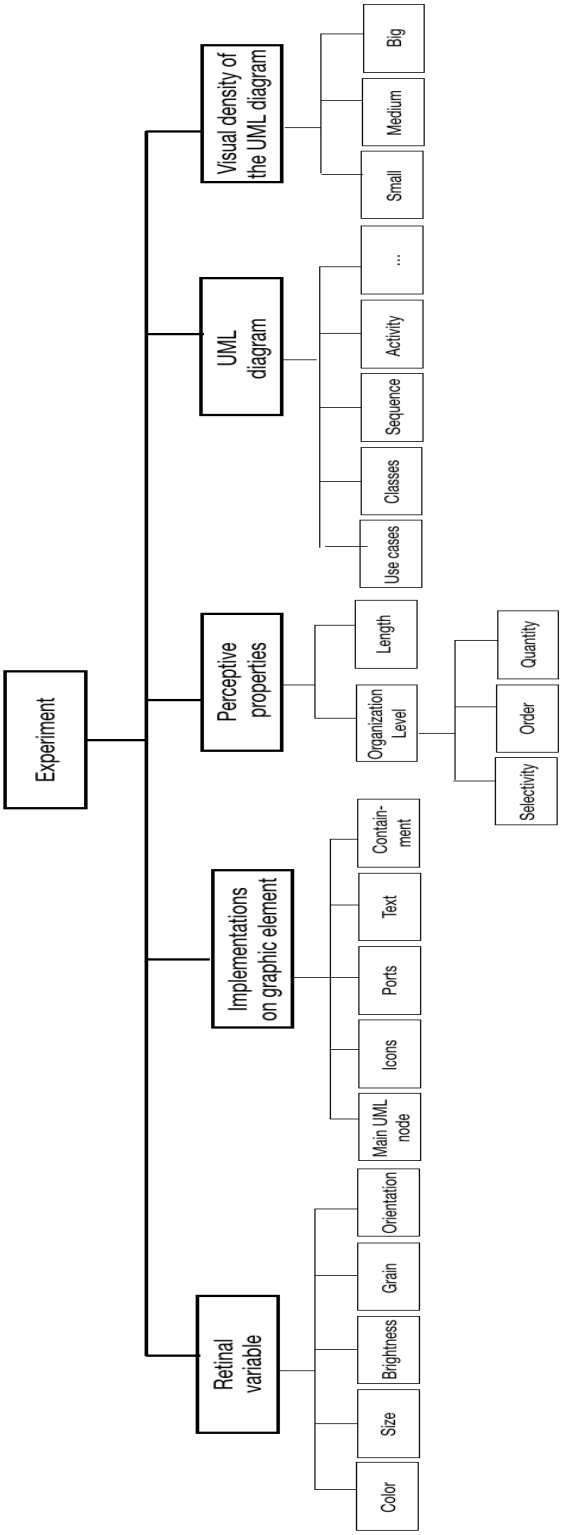

Figure 5.1: Scope of the empirical study.

5.1 Lessons learned from a conducted experiment

This section reports on an experiment which has been conducted with seventeen attendees of the HuFaMo'16 workshop [2]. The experiment aimed at finding the most effective implementation of the **size** retinal variable in **UML sequence diagrams**. Fourteen questions have been asked about fourteen different sequence diagrams. These diagrams were extracted from the Lindholmen database of UML diagrams [3]. They belong to sixteen different projects. For each diagram, four different implementations were tested. The experiment lasted around thirty minutes and each participant had to answer twenty questions. In the following two subsections we briefly describe the variables and the hypothesis of the study. An extended version of the design methodology is described in [19]. Then, we present the lessons learned from this experiment.

5.1.1 Variables of the study

The experiment had four independent variables. In fact, we tried to assess four different implementations (**Im**) of the size retinal variable to three different graphic components of the UML sequence diagram (i.e., message, lifeline and fragment). For each graphic component, we had one implementation, called *I*, that we deemed effective based on the SoG and three others that we judged less effective, called *I'*. We also varied the size (**S**) of the used sequence diagrams based on the number of components contained in them. They range from the small (*S*) ones by medium (*M*) to large diagrams (*L*). In this experiment, we also considered the distance between the linked elements to which we applied a size variation (**L**). For that, we considered the horizontal distance (*HD*) which concerns, for example, the distance between the source and destination lifelines of a message. We also considered the vertical distance (*VD*) which refers to the distance between the message and the name of the concerned lifelines, for instance. Finally, we asked two types of questions (**TI**): questions which concern one single graphic component (**TI1**) and those which concern more than one graphic component (**TI2**). The experiment had two dependent variables. The first one is the response correctness called (**R**) which might be *false*, *true*, *complete* or *incomplete*. Completeness here denotes if the participant has given all the elements of the correct answer or not. The second dependent variable is the response time called **T**. Below, the variables of the study are summarized:

Independent variables

Implementation **Im** (**alternatives**: Effective Implementation **I**, Other Implementations **I'**).

Size of the sequence diagram **S** (**alternatives**: small, medium, large).

Its layout (i.e., the distance between related graphic components) **L** (**alternatives**: Horizontal distance **HD**, Vertical distance **VD**).

Type of information to highlight **TI** (**alternatives**: concerns only one graphic component **TI1**, more than one graphic component **TIn**).

Dependent variables

Response correctness of the participants **R** (**alternatives**: true, false, complete, incomplete).

Response time of participants **T**.

5.1.2 Hypothesis of the study

The hypothesis for assessing the effectiveness of the **I** size variations with the independent variables are given in table 5.2. The alternative hypothesis H states that the proposed effective implementations take less time to let participants give the right and complete answer to a given question. The experimented effective implementation **I** is proposed for each possible combination of (S, TI, L).

Table 5.1: Hypothesis

Dependent variables	Null hypothesis	Alternative hypothesis
Response time T	∀ (S, TI, L); H0: T(I) >= T(I')	∀(S, TI, L) H1: T(I) < T(I')
Response correctness R	∀ (S, TI, L); H0: R(I)=false and incomplete; R(I')=true and complete	∀ (S, TI, L) H1: R(I)=true and complete; R(I')=false and incomplete

5.1.3 Lessons learned

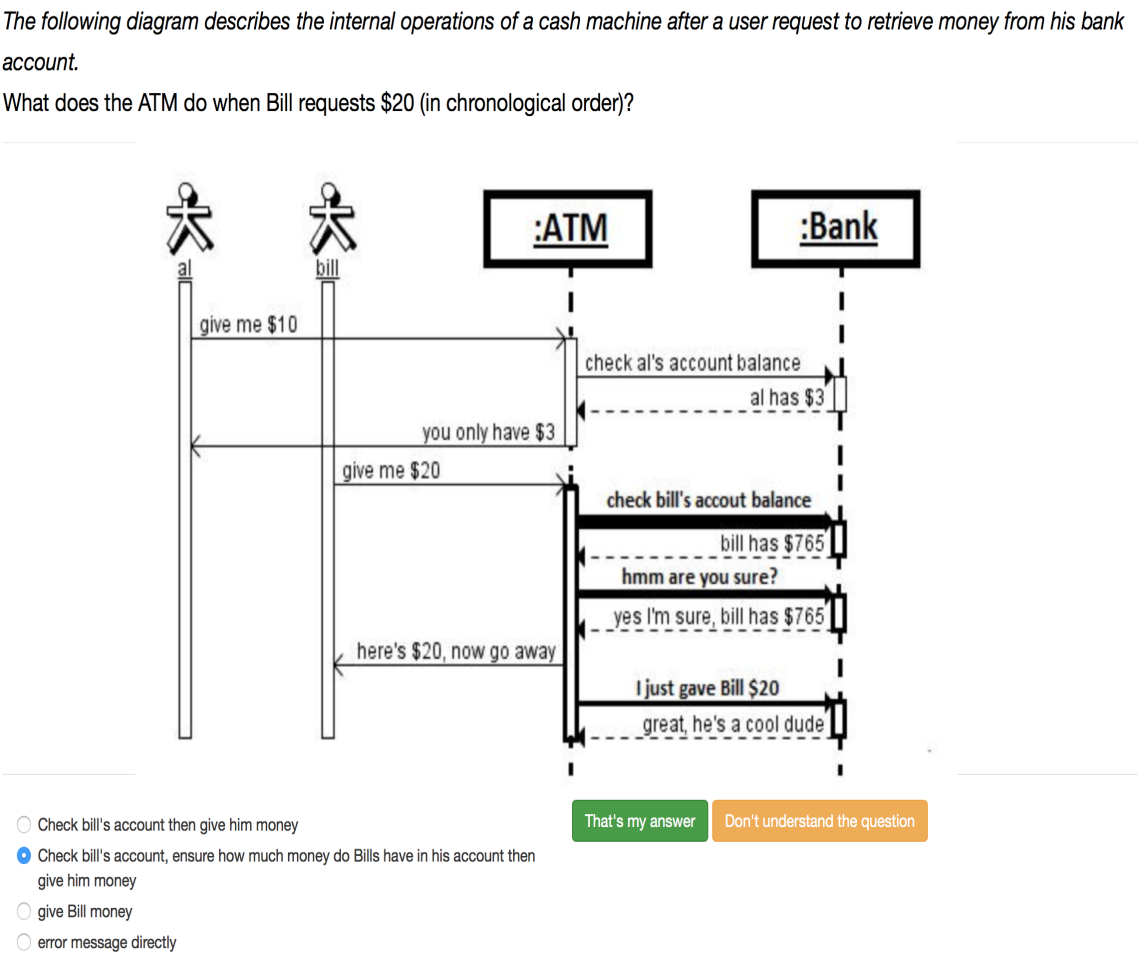

Figure 5.2: Experiment about the size variation on UML sequence diagrams: An example of a question.

We have seen the variables and the hypothesis of the experiment. In this section, we present the lessons learned after its execution and analysis steps. To better understand these lessons, Figure 5.2 above shows an example of a question, its related sequence diagram and the answer's options from the web application which has been used for the experiment.

Below, we explain the four lessons that we have taken from this study:

Ensure a good choice of the diagrams of the study

In the experiment, most of the sequence diagrams were related to the design of a modeling tool project. They were a big source of confusion to the participants. The sequence diagrams describe how to create UML diagrams using the modeling tool in question. To eliminate such confusion, the diagrams should be well chosen. They have to be simple and comprehensible. In addition, each participant worked on diagrams which belong to different projects. Switching from a project to another was difficult because the participants had to understand the context of each project. For that reason, questions should also belong to the same project.

Ensure that the questions are sufficiently comprehensible

The questions were difficult to understand. There was an option which allows the participant to indicate that she/he does not understand the question. Such option was checked many times in the experiment. An example of an asked question was: *"What are the two main behaviors of the system when users want to execute an update names?"*. It requires from the participant to remember three parameters: the number of behaviours (two), the object which triggers the event (the user) and the name of the event (update name). This type of questions was frequently asked, especially when it concerns more than one graphic component (**TI**=TIn). For that reason, the questions should be simple and comprehensible and they should be reviewed and tested many times before the experiment.

Ensure the accuracy of the collected response times

The web application displays the question in a textual form. The participants were meant to ensure that they well understood the question before clicking the button to display the sequence diagram, as illustrated in Figure 5.3 below.

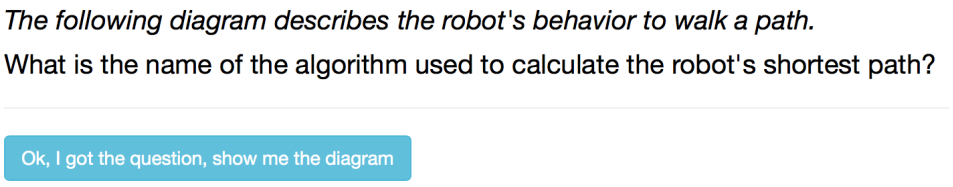

Figure 5.3: Reading the question via the web application.

Figure 5.4: Response time collection.

They also were meant to display the answers options after finding the correct an-

swer in the diagram, as illustrated 5.4. In that way, the web application can accurately collect the response times (i.e., the difference between the second and the first clicks). However, the participants displayed the diagram before even reading the question and the answers options before finding the correct answer. The response times were then biased by such actions. In addition, during the experiment, each participant used his own machine. Big diagrams did not fit to all screens. The participants had then to scroll down and/or right to search for the correct answer in the diagram. Such action also biased the response times, an important dependent variable of the study. For that reason, ensuring that big diagrams fit on all the participants screens helps accurately collecting the response times.

Use a large sample of participants and minimize the number of independent variables

It is clear that the number of participants was not representative of the target population. Such quantitative study requires around 100 participant to give statistically significant results. Indeed, the collected data from this experiment did not allow to give statistically significant results. Especially that most of the answers were "I don't understand the question". In that context, four independent variables were used in the experiment. With the small number of participants, each combination of these latter had a small number of answers. That did not allow us to decide about the most effective implementation of the size retinal variable for each combination.

5.2 Experiment definition

We took into account the previously identified lessons to define the following design methodology of experiment. For that, we present the research questions, the variables of the study and its hypothesis. Then, we provide the experiment design in terms of population sample, data collection material, the method, the data analysis procedures and the anticipated ethical issues.

5.2.1 Research questions

The effectiveness of an UML diagram is the key element that should be assessed in each experiment. The effectiveness of a graphical representation is defined as its capacity to allow the human reader to give the correct answer of a particular question. The answer should be given in a relatively short time compared to another less effective one [10]. In that context, the most effective implementations of a particular retinal variable should guarantee the highest level of effectiveness. The conducted experiment should then attempt to answer the following research questions:

RQ1: What is the most effective implementation of a retinal variable in a particular type of UML diagram?

RQ2: How is the effectiveness of an UML diagram controlled by the implementation of a particular retinal variable?

5.2.2 Hypothesis formulation

To answer the research questions, the experiment should have two independent variables and two dependent variables, as explained next.

Variables

Independent variables Implementation **I** (**alternatives**: Effective Implementation **I**, Other Implementation **I'**, Text via stereotypes **St**).
Visual density of the diagram **D** (**alternatives**: Small **S**, Medium **M**, Big **B**).

Dependent variables Responses of participants **R** (**alternatives**: true, false).
Response time of participants **T**.

Hypothesis

The hypothesis for assessing the effectiveness of an implementation **I** of a particular retinal variable with the independent variables are given in table 5.2. The alternative hypothesis H1 states that the effective implementation I takes less time to let participants give the right answer to a given question.

Table 5.2: Hypothesis

Dependent variables	Null hypothesis	Alternative hypothesis
Response Time T	∀ (D); H0: T(I) >= T(I') and T(I) >= T(St) and T(I') >= T(St)	∀(D) H1: T(I) < T(I') and T(I) < T(St) and T(I') < T(St)
Response R	∀ (D); H0: R(I)= false, R(I')= false and R(St)= false	∀ (D); H1: R(I)= true, R(I')= true and R(St)= true

5.3 Experiment design

5.3.1 Population, sample and participants

The target population of the study is composed of the users of UML including: software practitioners, researchers, students and modeling tools vendors. To find a representative sample of it, the convenience sampling might be used [15]. For that, the participants are not meant to be experts of UML. They should understand and know the UML diagram of the study. In fact, we mainly try to assess human factors in modeling and particularly vision (i.e., a factor that *all* the potential participants have). In that context, as we use a quantitative methodology, the higher the number of participants is, the more statistically significant are the results. The number of participants might be around one hundred participants (i.e., we tested the methodology with 95 participants and we had significant results, see Section 5.4).

5.3.2 Data collection and materials

To be aware of the complexity of modeling tools (i.e., the participants might not be familiar with the same modeling tool), a web application can be used in the experiments. The web pages might be displayed on a tactile tablet, to help accurately collecting the response times by eliminating the possible bias due to the non-familiarization with the used computers (i.e., by using a mouse instead of simply touching a screen). The

web application should first display an explanation page which describes the answering process. Then, it should ask the participants about their gender, level of experience in UML, if they have visual deficiency(ies) and if they wear glasses. Then, the experiment can start. The participant plays an audio question. She/he can listen to it many times to ensure its comprehension. She/he becomes able to click a button to see the diagram. An UML diagram which is visually annotated with an implementation of the retinal variable will appear. In parallel, the application triggers a time counter. The participant can then touch the elements of the diagram which belong to the correct answer. The application will save the position of each click. Therefore, in the analysis step, the answers correctness can be assessed. Finally, the participant clicks on a button to answer the next question. Thus, the application stops the chronometer and saves the time spent to answer. This web application has been developed and used in the context of this thesis (See Section 6.5).

5.3.3 Method

The experiment should be performed individually. The first step should consist of a brief announcement of the goal of the experiment to the participant. Then, the responsible of the experiment should explain the main treatment of the study which consists of the reading and the visual extraction of information from a visually annotated UML diagram. The mechanism of listening and answering the questions via the web application should also be clearly explained. The participant can better understand the latter mechanism by reading the explanation available in the first page of the web application. Then she/he can fill the form with information about her/him (i.e., gender, visual deficiencies, etc,.). After that, the participant should begin a training session on three questions concerning three UML diagrams. The latter training aims at familiarizing her/him with the web application. After that, the experiment can begin with fifteen questions about fifteen UML diagrams. The estimated time for the whole experiment is ten minutes. Finally, the participants answer a qualitative form at the end of the experiment. They have to rank the studied implementations, give their opinion about the use of the studied retinal variable in UML, if they already use it in practice or not and if the use of this retinal variable helps finding rapidly the correct answer.

5.3.4 Data analysis procedures

The analysis process should start by reviewing the accuracy of the participants answers. Then, only the correct answers should be taken into account in the analysis step. The *"false"* and *"no answers"* should be eliminated. They will serve later to judge the effect of each implementation on the the participants answers correctness. The ANOVA statistical test might be used to analyze the data. It allows the analyst to decide if she/he rejects or accepts the null hypothesis.

5.3.5 Threats to validity

This section reports on the internal and external threats to validity.

Internal validity

The first internal threat to validity is the possible gain of maturity by the participants during the study. That may happen because each question should be asked three

times (i.e., for each implementation). Therefore, the web application should ensure that diagrams will be randomly proposed, so that similar questions will not occur successively. In addition, as mentioned before, participants might have some visual deficiencies. This additional input will be mentioned before beginning the task and its influence on the results should be taken into account. Differences of the luminosity of different computers, the difference between the navigators, the screen sizes, etc., should be controlled in the experiment. For instance, the participants might use the same tablet in the same conditions: lighting, setting and surrounding noise. Finally, one of the outcomes of the study is the response time of participants. It is automatically saved when the participant finds the response by clicking a button. Late clicking the button will bias the results. The responsible of the experiment should stress on the importance of this step in the introduction phase. Finally, the questions should be simple and rapidly understood. For that, they must be reviewed and tested before the experiment. They should be written in the language of the participants (e.g., French for French participants).

External validity

The participants of the study will be around one hundred students, researchers and UML users. They correspond to a representative sample of the target population of the study. But, the participants are not in a natural setting, using their own modeling tool and moving *naturally* to their UML diagrams. To minimize such difference of setting, we propose that the questions should be inspired from the results of the previous empirical study (Chapter 2). The information properties which are expressed by the retinal variables can be the same as those mentioned by the practitioners. Such use might help *emulating* the situations of the UML practice (i.e., the questions concern the information properties that practitioners really needed to visualize in their practice).

5.4 Color and UML: A quantitative experiment

5.4.1 Methodology of the experiment

We have conducted an experiment which is based on the aforementioned methodology. The main objective of the experiment is to identify the most effective implementation of the color retinal variable on UML classes. In fact, the qualitative interviews with the practitioners and the quantitative analysis of thee UML diagrams (Chapter 2) showed that color is the most used retinal variable in practice. In addition, UML class diagram belongs to the three most used UML diagrams in practice (Chapter 2).

The experiment involved ninety five participants. Eighty of them are student from the University of Lille Sciences and Technology. The others are researchers on computer science from the Research center in Computer Science, Signal and Automatic Control of Lille (CRIStAL). They are decomposed as follows: seventy-four men and twenty-one women. Concerning their level of experience in using UML, we find that fifty-two participants have a medium level, thirty-eight are beginners and five participants are experts. They executed the experiment one by one, on the same tablet and on the same surrounding conditions. The variables, the hypothesis, the experiment definition and the anticipated ethical issues of the study are the same as defined earlier in Section 6.3.

Two different implementations were assessed in this experiment. The first implementation consists of the application of the color to the background of an UML class. Whereas, the the second implementation concerns the application of the color to a relatively thick border. Eventually, both of the implementations are compared to the the use of text via stereotypes. The experiment can be accessed via this link [36]. Figure 5.5 below shows the mechanism of answering the questions via the web application.

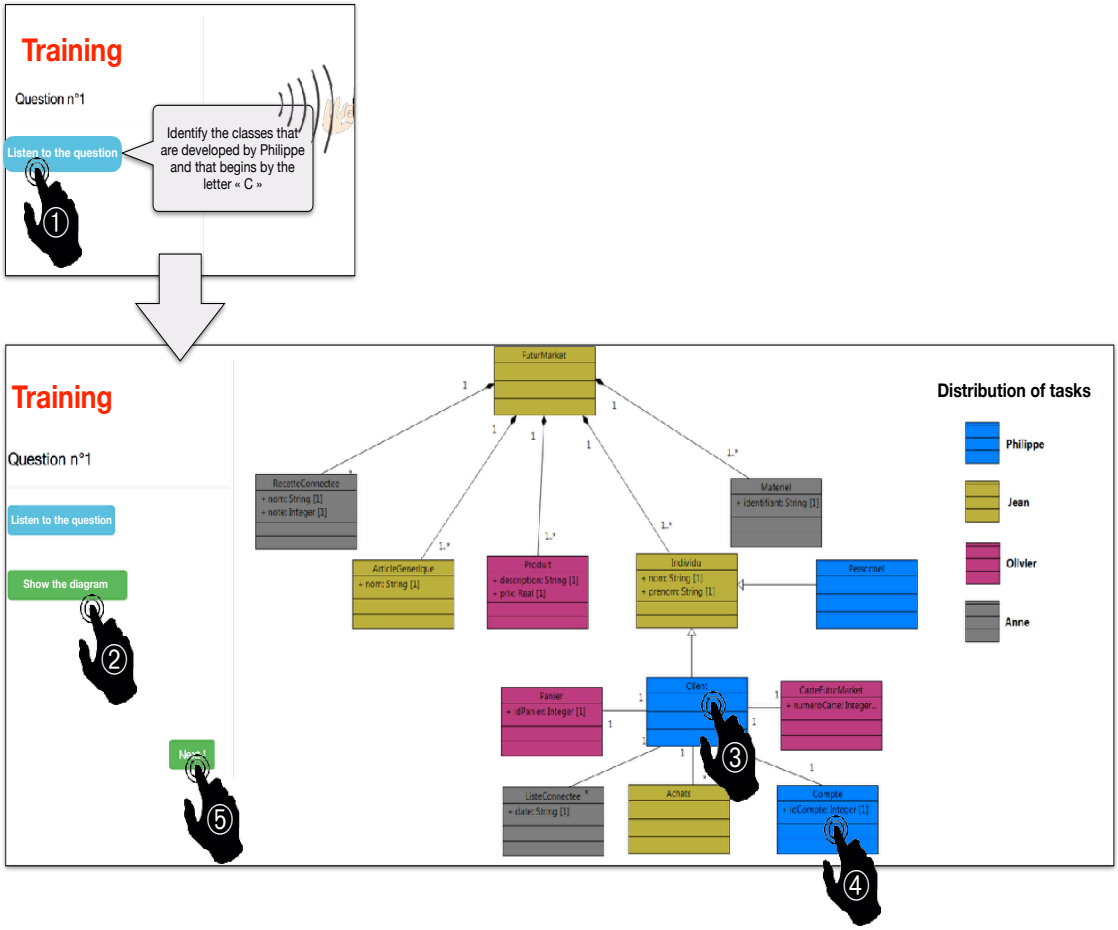

Figure 5.5: Experiments: the web application

Each participant had to answer three questions in the training session. Then, she/he had to answer fifteen questions about UML class diagrams describing a *pizzeria project*. The questions are detailed in the Table 5.3 below. We distinguish two types of questions. The first type of questions requires searching for a particular text in the UML class (e.g., a particular attribute/method, a particular characteristic of the UML class name). Such questions allow us to assess the influence of the coloration on the readability of the text. The second type of questions requires identifying elements which have the same color, there is no need to read text to find the correct answer. The information properties that were expressed in the diagrams were based on those pointed out by the practitioners in the previous empirical study (see Chapter 2). In addition, the questions relate to diagrams having different visual densities. Small diagrams contain three UML classes. Diagrams having a relatively medium visual density (i.e., compared to small ones) contain six or seven classes. Finally, diagrams having a relatively big visual density contain eight or nine classes.

Table 5.3: Questions of the study and the size of the related diagrams.

Questions of the experiment	Information property	Type of the question	Size of the diagram
Identify the classes having a level of criticality that equals 1 and that their name begins by "C"	Criticality	Text	Big
Identify the classes that are responsible of the access management and that their name begins by "A"	System's concerns	Text	Big
Identify the classes that are not members of the inheritance relationship and that are not enumerations	Inheritance relationship	Text	Medium
Identify the classes that are developed by Sébastien	Distribution of tasks	Class	Medium
Identify the classes that are non-members of the inheritance relationship	Inheritance relationship	Class	Medium
Identify the classes that are not members of the inheritance relationship and that contain the attribute named "Password"	Inheritance relationship	Text	Small
Identify the classes that are not yet tested and that contain the method named "Subscribe"	Testing progress	Text	Small
Identify the classes that are responsible of the administrative management	System's concerns	Class	Small
Identify the classes that are developed by "Sophie" and that begin by "M"	Distribution of tasks	Text	Big
Identify the classes that are responsible of the commands management	System's concerns	Class	Big
Identify the classes having a level of security that equals 3	Levels of security	Class	Big
Identify the classes that are developed by Mathilde and that have at least one relationship with the classes developed by "Quentin"	Distribution of tasks	Class	Big
Identify the classes that have a level of criticality equals 3	Criticality levels	Class	Medium
Identify the classes that are developed by Philippe and that contain the attribute named "email"	Distribution of tasks	Text	Medium

Note that the same range of colors has been used for both implementations (i.e., background and border): a set of selective colors having the same level of brightness (yellow, gray, blue and purple). The position of the keys did not change for all diagrams: in the right upper corner. That aims at eliminating the possible influence on the dependent variables of the study.

5.4.2 Analysis

We have seen the design of the study. In this section, we present the obtained results about each dependent variable (i.e., answer correctness R and response times T). Then, we illustrate the results of the post-experiment questionnaire.

Analysis of the answers correctness

Figure 5.6 below illustrates the answers of the participants per implementation (i.e., colors on the background, on the border or the use of text via stereotypes). The implementation of colors on the background of UML classes helped the participants to find the biggest number of correct answers (i.e., 431 correct ones) and, in parallel, the lowest number of false answers (i.e., 37 false ones). Whereas, the implementation of colors on the border of UML classes (416 correct answers against 47 false ones) is ranked second and the use of stereotypes is the last one (411 correct answers against 47 false ones).

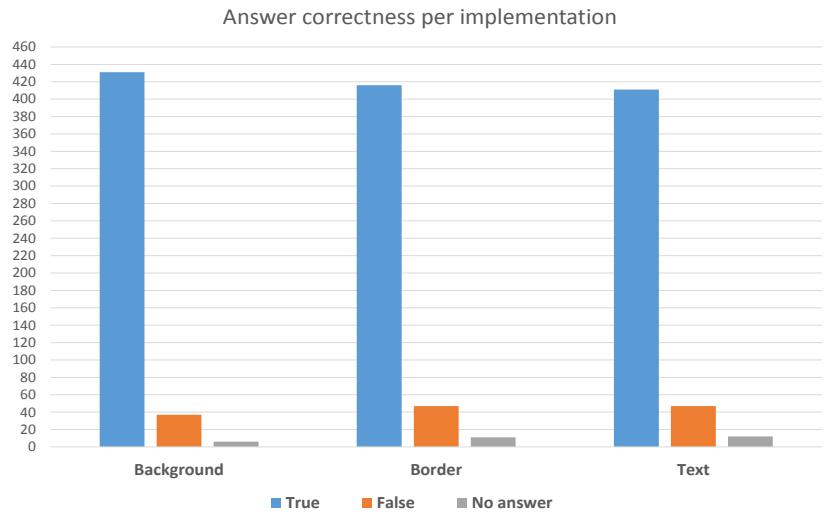

Figure 5.6: Answer correctness per implementation.

Analysis of the response times R

At this step, we eliminated the *false* and *no answers* from the data. Then, we executed the ANOVA statistical test on the reminder of the data using R. Repeated-measures ANOVA revealed a significant effect of *Implementation* on the *Response time* ($F_{(2,188)}$=12,21, p<0.05, η^2=0.04). Pairwise comparisons showed a significant p-value (p< 0.05) between the implementation which consists of using text via stereotypes and both other implementations (i.e., border and background). There is a non significant difference between the implementation of colors on the background and on the border. The mean response times of respectively the background implementation, the border implementation and stereotypes are: 9024.132ms, 9638.822ms and 10999.26ms (see Figure 5.7 above). We also assessed the effect of the size of the diagrams and the types of the questions on the response times. We found out that there is non significant

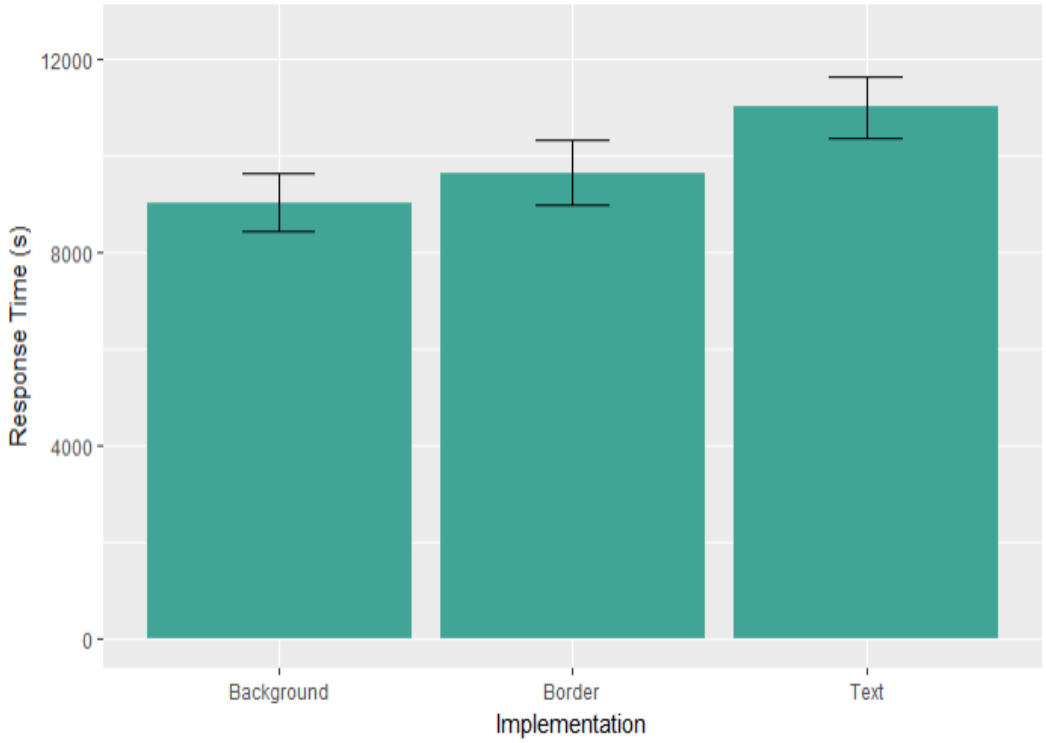

Figure 5.7: Box-plot: the results of the study.

($p>0.05$) effect.

These results allow us to reject the null hypothesis and conclude that, as recommended by SoG-UML, an effective implementation of a retinal variable, here colors, helps having effective UML diagrams. The effectiveness is here illustrated via the big number of correct answers and the reduced response times related to both implementations of colors. The non significant difference between both implementations of colors is explained in SoG-UML, as stated by our following guidelines:

> **Colors should be applied to the background of an UML node.**

> **The thicker the border and separation lines are, the better colors are visually selected.**

We used colors on a relatively thick border of UML classes, which represents an effective implementation along with the implementation of colors on the background. In addition, coloring the background of UML nodes has no effect on the response times. The use of effective colors in the experiment (i.e., medium level of brightness) does not influence the readability of the text of the UML classes, as confirmed by our following guideline:

> **Colors in medium levels of brightness should be used for UML nodes. They ensure the readability of black text labels, borders and separation lines and provide the biggest number of selective colors.**

Analysis of the post experiment results

The participants were asked to fill a post experiment questionnaire. They had to rank the six following parameters from one to five (i.e., *one* means not good, *five* means: excellent):

- The effect of using colors on the answer correctness.
- The effect of using colors on the rapid identification of the correct answers.
- Their preference in terms of implementations of colors: borders or background.
- Their preferences between the use of colors and the text via stereotypes.
- Their opinion about the use of keys on understanding the color variations.
- Their current usage of colors in UML.

Figures 5.9 and 5.8 below illustrate the participants answers of the post experiment questions. We divided the answers on those having a rank lower than three and those higher than three in Figure 5.9. In that context, we notice that most of the participants find that the use of colors helped them to not only find the correct answer but also in a shorter time. They also find that the use of the keys when colors are used facilitates understanding the color variations.

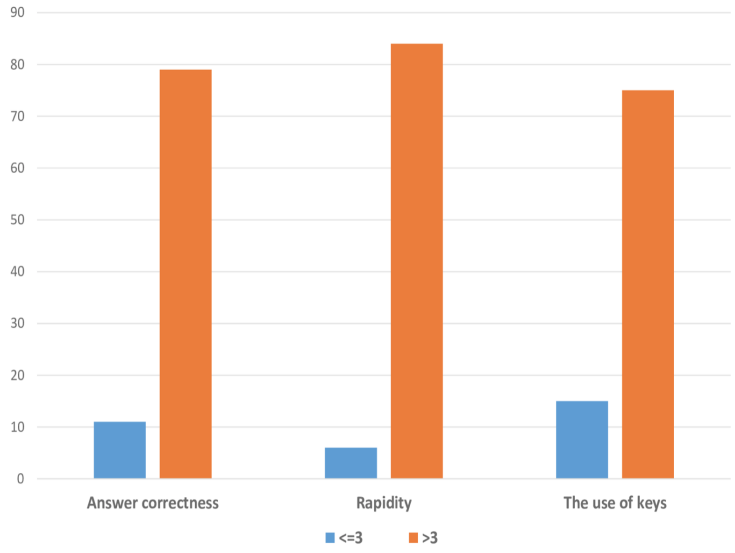

Figure 5.8: Post experiment: Participants opinions

Concerning their preferences of implementations, we find that most of the participants would prefer applying colors to the background of UML nodes. In addition, most of them would prefer using color than text after executing the experiment while they were not used to employ them in practice.

These results reinforce the SoG-UML guidelines as the participants find that the use of colors helps identifying the correct answers in a rapidly short time, the definition itself of an effective graphical representation [10]. We note that two participants were *daltonien* but found that the used colors were pretty distinguishable for them.

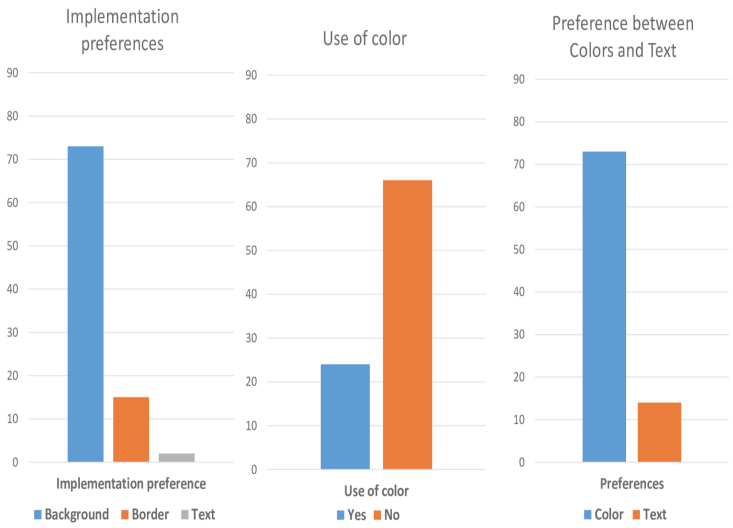

Figure 5.9: Post experiment: Participants opinions.

5.5 Summary

In this chapter, we presented a design methodology of an empirical study, having an experiment as a strategy of inquiry, to assess the SoG-UML guidelines. This methodology takes into account the lessons learned from a conducted but failed experiment in the same objective. We executed the proposed experiment to assess the effective implementation of colors on UML classes. Results showed that three guidelines about colors in SoG-UML are empirically validated. They also showed that the use of colors helped most of the 95 participants to rapidly find the correct answers, compared to the use of text via stereotypes. The results of the post-experiment questionnaire strengthen also the SoG-UML guidelines. Most of participants find that colors were helpful and they might use them henceforth.

The present design methodology is complementary to the theory proposed by SoG-UML. As SoG-UML is based on the *design and action* theory, it should include methods to assess its defined guidelines. If Moody [34] settles for giving the dependent and independent variables of a possible empirical study to assess his principles, we define a complete and validated design methodology. We also validate three guidelines of SoG-UML.

Chapter 6

Implementations

Contents

We have defined SoG-UML with a validated experimentation protocol. In this chapter, we describe two prototypes of tools, as proofs of concept. They aim at structuring the use of the visual variables in modeling tools, particularly in the Papyrus modeling environment. In that context, the following question arises: How are these visual variables actually presented: from the standard by modeling tools to users?

The OMG has standardized the bridge between the UML specification and modeling tools via the Diagram Definition framework (DD) [35]. DD specifies the mapping between the UML abstract syntax and its concrete syntax. Its architecture distinguishes between the visual variables depending on whether UML practitioners have control over them or not. DD identifies the color, size, orientation, brightness and grain as auxiliary visual variables that might be interchanged between modeling tools.

Modeling tools are free on how UML practitioners access the visual variables. To do that, they currently use property views, toolbars or CSS style-sheets [33]. UML practitioners are also free to use these visual variables, when access is given by tools. For that, they might rely on their own taste, preferences, culture, etc. They are also free to define internal conventions within their team (e.g., the green color of a port means that it is an input port). The freedom given to both tool-vendors and UML practitioners might be disadvantageous insofar as no rules of effective use of the visual variables are defined and/or integrated in modeling tools.

Both of our tools try then to fill this gap in Papyrus. The first tool, called Fliplayers, introduces the concept of layers to UML modeling tools. The second tool is a continuity of FlipLayers. It consists of an interactive keys for UML diagrams and provides a simple and rapid way to effectively visually annotate UML diagrams. Fliplayers is released in the Papyrus modeling environment [1] and is under work for further enhancements [5]. The interactive keys is still in the development step.

6.1 FlipLayers

FlipLayers is an implementation of the classical concept of layers (e.g., image manipulation programs) in UML diagrams. In this section, we provide a general overview of FlipLayers and its capabilities.

6.1.1 FlipLayers Graphical Mechanism

A diagram is one particular view of an underlying model. That is generally the reason why multiple diagrams are needed to represent a system model where each one is focusing on one or several concerns. The elements of a diagram consist of two kinds of data: graphical data and domain (model) data. From the SoG viewpoint, graphical data consists of values assigned to the visual variables used by the background, the border and the text of labels. This kind of data can be modified by setting graphical properties of the element. We make a difference between visual variables and graphical properties because the graphical properties sometimes mix several visual variables (like the Red-Green-Blue value which is the addition of a choice of a primitive color and a brightness level). Note that modifying any model element (e.g., its name), in one diagram may have to be reflected in other diagrams depicting that same element. In contrast, modification of a graphical element (e.g., position, size, color), will a

priori impact only the diagram in which the change was made. A layer is a kind of

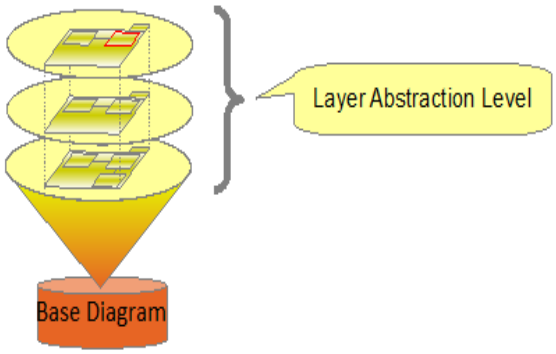

Figure 6.1: The graphical layer mechanism

filter and/or selector that can be overlaid on a diagram. *Layering* enables selective control of the graphical properties of the graphical elements of a diagram. The layers mechanism consists of a stack of layers which are superimposed on an underlying diagram, called the base diagram, as illustrated in Figure 6.1. Each layer controls the graphical properties of a chosen subset of domain elements of the base diagram. The current version of FlipLayers does not support all the Bertin's visual variables (Section 5.1.6 describing these limitations). On the other hand, the current visual variations proposed by FlipLayers provides enough material to create numerous types of views.

6.1.2 Layer Stack

Layers can be placed on top of other layers in what is called a stack. A stack plays a dual role: it allows superposition of visual information of individual layers, but it also allows users to define how the layers are to be combined, thanks to a variety of stack operators (e.g., and, or, xor). When a diagram is viewed through a stack, the resulting view is a combination of the representations of individual layers based on the values of these operators. This computation proceeds from the top layer downwards: for each graphical element, an imaginary ray is "launched", starting from the top layer and penetrating through the layers, asking each layer in turn if it controls the graphical element. If so, the values of the graphical properties attached to layer are collected. If there is only one value for a graphical property, then this value is applied to the matching graphical element in the base diagram. If multiple values are collected for a given graphical property, then the stack operator is used to calculate the resulting values. A layer stack can itself be seen as a layer: the attached graphical elements are the union of the graphical elements attached in layers, and graphical properties of each graphical element are those calculated by the stack.

6.1.3 FlipLayers Operators

We have defined several stack operators in FlipLayers: the operator *top* simply holds the first value encountered, that is to say the first value seen when looking at the layers from the top. This is the default operator. For Boolean values, the *and* and *or* operators perform a logical conjunction and disjunction operations respectively, and a *top* operation for other types of values. We have also experimented with the operators *average*, *min*, and *max*, which compute an average or enforce the minimum or maximum

value for numerical values. The average operator allows visualizing intersections of sets: each set of graphical element is attached to a layer thanks to an OCL expression. Each layer shows its set with a different color.

6.1.4 FlipLayers Graphical Properties

Each layer associates a set of graphical variables whose values can be used to express different information properties. The variables that are currently available are:

- *Visibility.* This allows selectively showing or hiding some elements of a diagram. Hiding an element can help to reduce the visual complexity of large diagrams.
- *Fill color* This allows changing the background color of the UML elements, of the border of these ones and also of their corresponding text.
- *Brightness.* This variable works in conjunction with the color variable; i.e., after choosing a particular color, users can change its tone for each layer. Similarly to the color, it affects backgrounds, borders and also text.
- *Line thickness*
- *Text font*

The graphical elements can be attached to a layer in two different ways: a) by explicitly listing the graphical elements associated with the layer. b) by specifying an OCL expression, which selects the subset of graphical elements in the base diagram that are controlled by the layer. This form can be quite powerful, as it allows complex expressions such as "all classes whose name begins with 'RT' ", "all interfaces in which at least one of the operations is named 'execute'", "all elements stereotyped 'priority' whose property 'value' is equal to '3' " and so on. The user must also specify which properties are attached to a layer. It is possible to associate only a subset of possible graphical properties with a layer. As a result, only the graphical properties explicitly associated with a layer can be set within that layer. A given graphical element can be attached to several layers simultaneously. This allows, for example, to specify its color in one layer, its font in another, and its visibility in a third. The user will then see the graphical element decorated with the merge of graphical properties defined for each layer using the algorithm described earlier.

Example of use of FlipLayers

The principle of using FlipLayers is as follows. The person who wants to create one or several views starts by attaching a layer stack to the current base UML diagram. Then, she/he creates a layer or several layers on top of this layer stack. She/he can assign meaningful names to these different layers. Next, she/he selects the UML elements that she/he wants to group together and appends them into one selected layer. At this point, the author can modify the value of the different visual variables according to the information property that she/he wants to express through the representation.

We consider a use case diagram of the Internet Banking System (IBS). We express the project progress information property using Fliplayers, as shown in Figure 6.2. We see that there is one layer for each group of use cases. Each layer has the color property attached. After setting the values for each layer, we obtain the sum effect

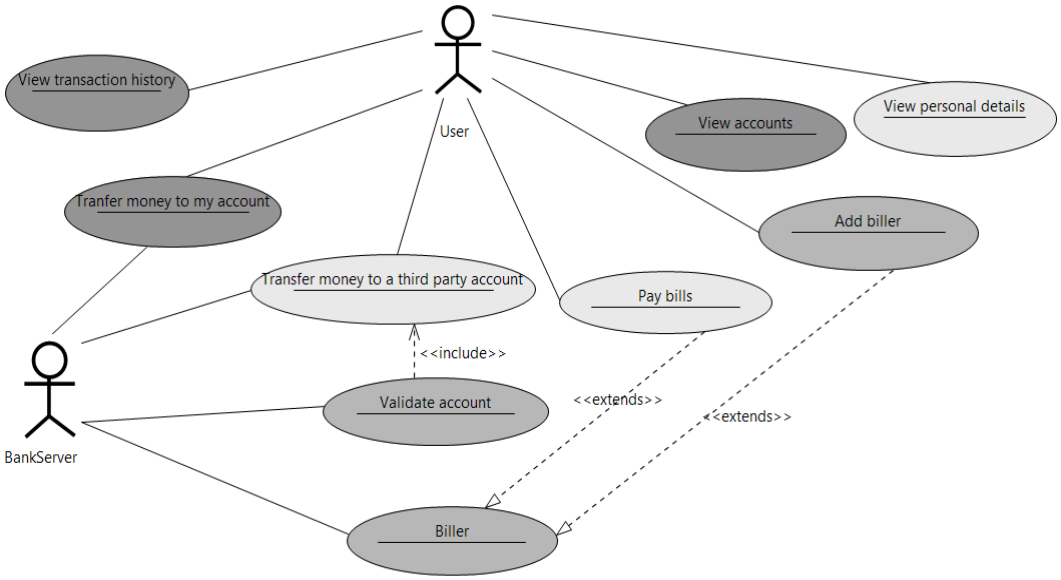

Figure 6.2: Using FlipLayers to express the project project progress in the IBS use case diagram.

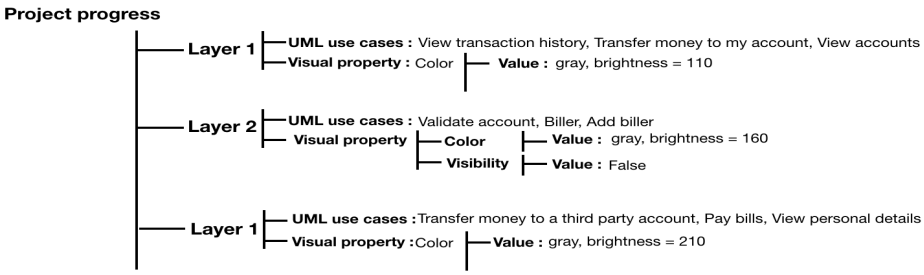

Figure 6.3: A layers stack: Project progress criteria.

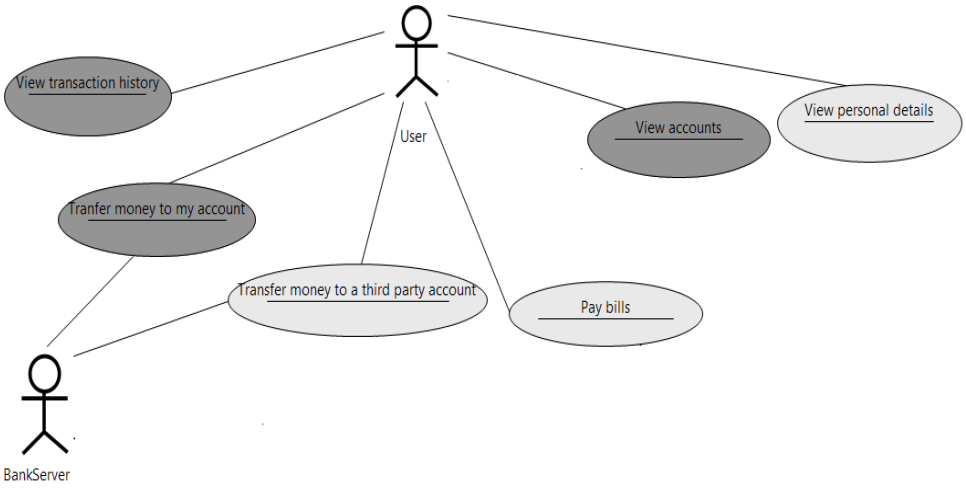

Figure 6.4: The visibility property of FlipLayers

shown in Figure 6.2. The visibility property might be attached to each layer. It allows controlling its embedded elements, by making them visible or not. Figure 6.3 above illustrates the layer tree of the use case diagram shown in Figure 6.2. The visibility property is attached to Layer 2. Hence, users might make the corresponding use cases visible or not, as illustrated in Figure 6.4.

Actors of FlipLayers

The main actors of FlipLayers are the Papyrus modeling tool users. They can create graphical layers on top of their UML diagrams and are able to define different types of relationships between them (e.g., and, or and xor). An actor of FlipLayers can be a project manager who prepares a communication with his client or within his technical team. He might also be a software designer who wants to save the semantically linked elements between the UML diagrams of his model. Such information can help navigating between the UML views to facilitate later maintenance tasks. Finally, an actor of FlipLayers might be a software engineer who wants to build his own understanding of the modeled system or to create documentation. Creating and manipulating layers might be considered as time consuming for UML practitioners. But, practitioners can earn time while using the resulting diagrams via better communications, better manipulations of large diagrams and via easier navigation between UML views.

6.1.5 Summary and discussion

A variety of graphical properties are available in FlipLayers, such as fill color, border size, font of texts. One of these graphical properties is visibility, which can help hiding elements of the base diagram and thus reduce the size and complexity of large diagrams. All of these properties are used to convey meaning to human readers and to express their different information properties. An example of using FlipLayers on the IBS use case diagram was described. The use of FlipLayers and the retinal variables might facilitate stakeholder communication via meaningfully annotated UML diagrams. Such approach primarily addresses readability issues associated with UML diagrams. It provides a general mechanism to improve the communication value of UML and to better exploit UML for communication. One of the main expected results of such approach is to reduce the required cognitive effort to understand complex diagrams. It serves to highlight specific concerns in a given diagram, as well as to handle cross-cutting concerns that may be shared by multiple diagrams (e.g., assigning the same color to the semantically linked UML elements). But, FlipLayers presents some weaknesses. It does not provide the retinal variables separately as recommended by the SoG (e.g., mixing colors and brightness). In addition, it does not guarantee the effectiveness in using the retinal variables. For instance, it does not automatically generate keys when at least one retinal variable is employed. The interactive keys comes then to cater for these incompleteness in FlipLayers, as described in the next section.

6.2 Interactive keys for UML diagrams

The interactive keys is an Eclipse plug-in integrated in the Papyrus modeling environment. The main objective of this tool is to provide a simple, rapid and effective way to use the retinal variables for the UML practitioners. To that end, it allows them to simply assign *tags* to their UML diagrams. The visual annotation of the UML components (i.e., putting some components in a third dimension) is then done by giving values to these *tags*. By *tags*, we refer to information properties that an UML practitioner wants to express in her/his diagrams (e.g., project progression) and their values mean their categories (e.g., done, in progress and to do are the values of the project progression tag). The interactive keys also assists the practitioners in choosing the effective retinal variables and their categories based on the characteristics of the corresponding tags.

Finally, it plays the role of the keys/legends or captions. It helps reading the visually annotated UML diagrams by displaying the meaning of each retinal variable variation. Below, are described the functionalities of the interactive keys.

Visual annotation: a simple tag of UML components

The interactive keys enables UML practitioners to simply define new tags or use existing ones in their UML diagrams. For that, they only have to enter the name of their tags (e.g., a tag named *distribution of tasks*). Then, they have to simply select the UML elements (e.g., use cases, classes, activities) that share the same value of the tag (e.g., the classes that are developed by one developer named *John*) and assign this value (i.e., John) to them. As illustrated in Figure 6.5 below, the UML practitioner wants to express the *implementation progress* in his use case diagram. For that he selects the use cases which have the same level of progression and tag them with their corresponding level of progress (here *done*).

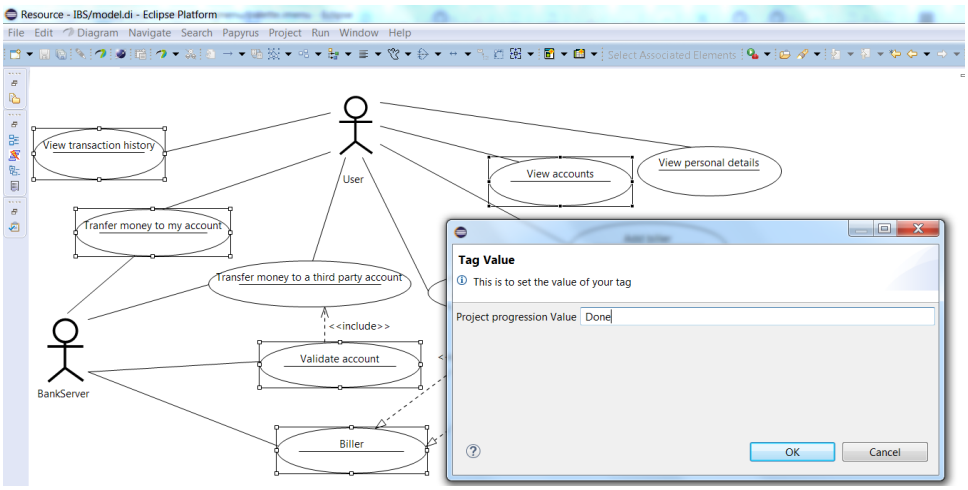

Figure 6.5: Visual annotation process using the interactive keys.

Each new tag will be automatically added to an UML profile which is attached to the corresponding UML diagram. In this profile, the plug-in saves all the tags (i.e., as stereotypes) and their categories. It is dynamically updated and re-applied to the diagram in question when a new tag or category of an existing one is added. In that context, the attached profile just extends the Meta-class *Element* to support the elements of all the types of UML diagrams. Then, for each new tag, the plug-in creates a new stereotype which extends the meta-class *Element*. Thereafter, it creates an *Enumeration* which will contain the categories of the tag in question as *EnumerationLiterals*. The stereotype will then contain an attribute with that enumeration as type. This automation helps fastening the visual annotation process for the practitioners.

Interactive keys: a rapid and an effective visual annotation using the retinal variables

When a practitioner enters the name of a tag, she/he should mention the characteristic of this one by simply choosing its organization level from a scroll-able list (i.e., selective/nominal, ordered or quantitative). As a result, the interactive keys will propose the possible effective retinal variables and omits those which do not suit with the tag.

For example, if the tag is ordered, the interactive keys proposes the size, the brightness and the grain. But, the color variable is omitted, as shown in Figure 6.8 below. In addition, for each each proposed retinal variable, a set of effective categories is given. For example, a set of selective colors which have the same level of brightness is proposed as ready-to-use categories. Also, the effective implementations, chosen by SoG-UML, are directly provided (e.g., the colors and brightness to the background). The interactive keys plug-in creates automatically a layer using FlipLayers. Thus, the visibility option still be available.

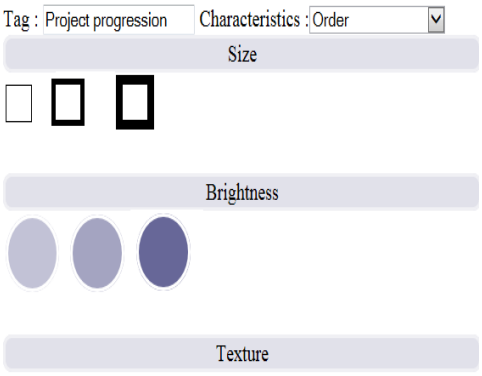

Figure 6.6: The interactive keys proposition for an ordered tag: Project progression.

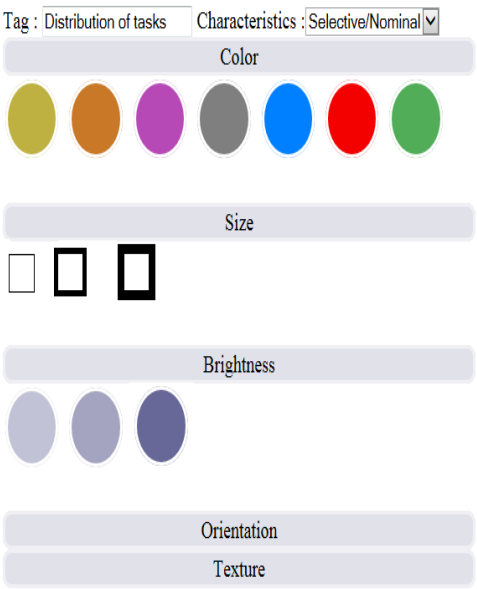

Figure 6.7: The interactive keys proposition for a selective tag: Distribution of tasks.

Figure 6.8: Assistance to the effective retinal variable use.

The interactive keys as captions or legends

The interactive keys allows displaying the mapping between each category of a retinal variable and the category of the tag it expresses. It plays the role of the captions/legends. Figure 6.9 shows the resulted captions from the previous visual annotation processes. The corresponding levels of the tag *Project progression* are displayed below each level of brightness in the keys.

A practitioner can express another tag in the same diagram by using a different retinal variable. Figure 6.10 illustrates an example where the *Project progression* and the *Complexity* tags are expressed using respectively the brightness and the size retinal variables. By observing this visual annotation, a project manager can see that there is only one complex functionality which is finished: *View transaction history*. However, two complex functionalities are still not yet done. *Transfer money to a third part account* is not even begun and *Pay bills* is in progress. He might then rethink the

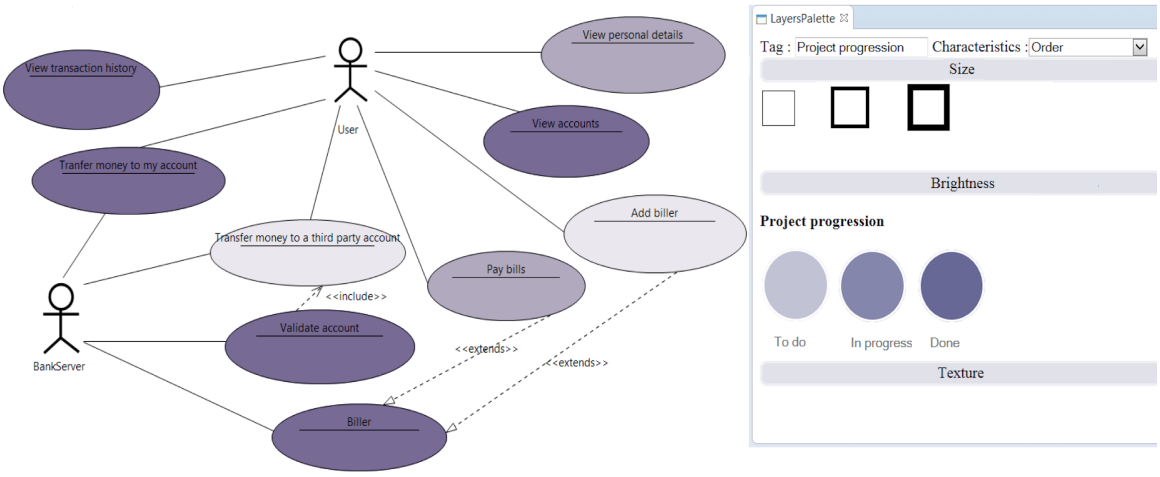

Figure 6.9: The interactive keys as captions or legends.

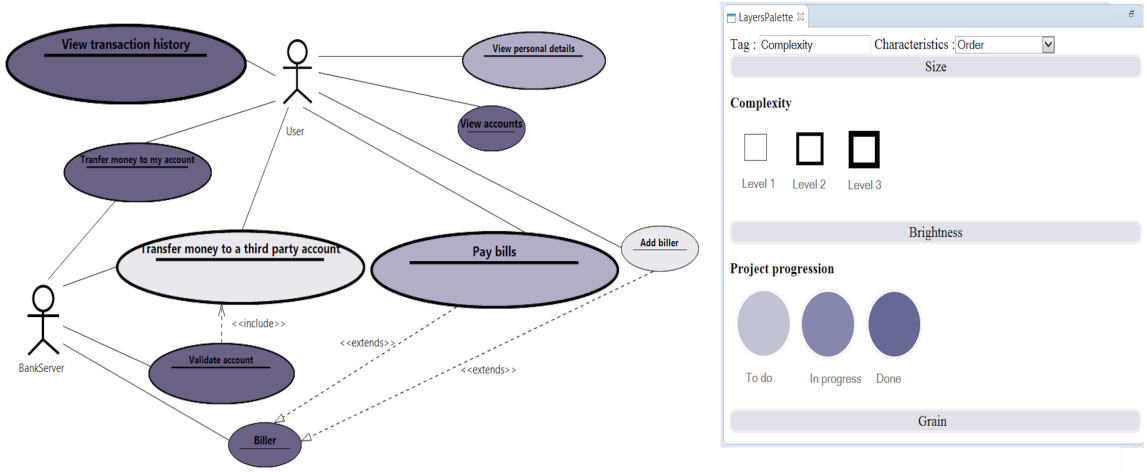

Figure 6.10: The interactive keys as captions or legends: two tags.

distribution of tasks to meet the approaching delivery day. The visibility option might be enabled to visualize only one tag. In that context, the grain retinal variable might be used instead of the size variation. However, this kind of variation is not yet available in our plug-in, as stated in the next subsection.

6.2.1 Summary and discussion

The interactive keys aims at providing a simple and a rapid way to visually annotate UML diagrams using the retinal variables. It allows an UML practitioner to focus only on the concepts represented by the tags that she/he wants to express and makes abstraction of the effective use of the retinal variables, which is handled by our plug-in. Below is summarized the simple workflow of using the interactive keys:

1. Enter the name of the tag and its characteristics.
2. The interactive keys proposes the retinal variables which are suitable to the tag.
3. Begin the visual annotation:
   (a) Select the UML elements which have the same value of the tag.

(b) Click on the corresponding category of a proposed retinal variable.

(c) Enter the value of the tag.

(d) The interactive keys displays this value under the corresponding category of the retinal variable.

4. Go to *3* until completing all the categories of the tag.

5. The interactive keys displays the full captions.

6. Go to *1* for a new tag.

This proof of concept tool requires further improvements. For example, the grain and the orientation retinal variables are not yet implemented. Their variation is difficult to construct in the existent technical configurations. In addition, the keys should be generated and embedded to each diagram. Also, it should further integrate the recommendations of the practitioners in the previous empirical study (in Chapter 2). For example, the dynamic variation of the values in the keys and the diagram can be implemented.

Chapter 7

Conclusion

Contents

7.1 Conclusions

The main objective of this thesis was to enhance the cognitive effectiveness of UML diagrams by taking advantage of the following five retinal variables: color, size, brightness, grain and orientation. We began our research by an empirical study to deeply understand the actual state of practice in using UML and these retinal variables by the UML practitioners [20], persons for whom this work is mainly conceived.

Results showed that UML diagrams are employed in different situations where practitioners need to visualize information valuable for their tasks. In order to highlight these information, practitioners sometimes rely on colors and they find that the use of color helpful. The other retinal variables (i.e., size, brightness, texture/grain and orientation), are not actually employed in practice but most of the practitioners think that they might be useful. This usefulness is depends on the usability of modeling tools in terms of rapidity, intelligence and dynamism. Subtlety in using these retinal variables is also recommended by the necessity of attaching a meaning to each visual variation. In the +3500 UML diagrams, color is also the most used retinal variable whereas the size and brightness are less employed. The analysis of the UML diagrams revealed their recurrent non-effective use. For instance, only four per cent of the diagrams had keys when colors were used and sometimes they were not updated with the corresponding diagrams. Moreover, the retinal variables were differently applied to the border, text, background or compartments of UML nodes. Some of these implementations are more effective than others (as discussed in Chapter 4). Finally, sometimes there was a non effective use of colors via non effective categories or via using color to express ordered information.

After the previous empirical investigation, we studied the existing works in the literature which deal with the cognitive effectiveness of programming environments: the cognitive dimensions framework and the physics of notations framework. We have shown that the CDs framework represents an only evaluation framework by applying its 13 dimensions to the three most used UML diagrams in practice (i.e., use case, class and sequence diagrams). Then we presented the PoNs framework. We showed that, although it represents a good scientific basis to build effective visual notations and to evaluate existing ones, the principles cannot be used directly [42]. Research theories were synthesized (i.e., including the SoG) which led to a certain incompleteness in some principles. We chose then to study the SoG which is the reference of the cartography in the context of UML which is the most widespread visual modeling language in software engineering. That aims at providing explicit guidelines to lighten the cognitive load in some dimensions of the CDs framework and some principles of the PoNs framework.

As a first refinement layer to adopt the SoG principles to UML, we presented the main notions of the SoG along with a parallel positioning against UML. The visual variables and their perceptive properties, the analysis and reading processes and the characteristics of each retinal variable were discussed in the UML context. We particularly discussed the rules of the SoG which might be directly applied to UML. But at that stage, we observed that the SoG principles cannot be directly mapped to UML because of the graphic complexity of the UML concrete syntax which was not explicitly addressed by the SoG.

For that, we empirically classified the UML concrete syntax and found out that UML nodes are complex because of multiple facts. They are mainly composed of shapes which might contain text labels, compartments, headings, icons and ports. They can also be contained in each other (e.g., a class in a package). Then, for each retinal variable, we were faced to the following problem: How can we handle such graphic complexity and, in parallel, guarantee a better cognitive effectiveness of UML diagrams by taking advantage of the retinal variables?

SoG-UML answers this question and defines 61 guidelines to visually enrich UML diagrams. It relies on the design and action theory to describe explicit rules about the effective use of the retinal variables in UML nodes. In SoG-UML, for each retinal variable, we justified the choice of its most effective implementation on the groups stemming from the previous classification, including the main UML nodes and their inner constituents, the containment's relationships and the combination of the retinal variables. We illustrated the SoG-UML guidelines via the diagrams related to an Internet Banking System IBS which was extracted from the Lindholmen database of UML models [3].

To make of SoG-UML a testable theory, we presented a design methodology of empirical study which serves to validate the SoG-UML guidelines. For that, we described the lessons learned from a conducted experiment which did not succeed to give statistically significant results. Then, we presented the design methodology of an empirical study which was tested and succeeded to give significant results. It consists of a quantitative methodology based on an experiment as strategy of inquiry. It was executed with 95 participants from the University of Lille and the CRIStAL laboratory. The main objective was to validate the effective implementation of the color retinal variable on UML classes. Results showed that the use of colors, using the two tested implementations, helped the participant to find the correct answer in a relatively short time compared to the use of text via stereotypes. We tested the implementations of colors on the background and on a relatively thick border of UML classes. Both implementations were recognized as effective in SoG-UML and were validated by the study.

Finally, we described two proof of concept tools which are integrated in the Papyrus modeling environment. The first tool consists of an implementation of the classical mechanism of graphical layers in UML diagrams. It provides the possibility to control the visibility of a set of UML nodes together. Such functionality allows reducing the complexity of big UML diagrams. In addition, combining such functionality with the retinal variables might help enhancing communications with UML by supporting the user's discourse. The second prototype of tool consists of an interactive keys for UML diagrams. It provides a simple, rapid and effective way to exploit the retinal variables in UML. It allows practitioners to simply assign tags to their UML components and assists them to an effective choice of the suitable retinal variables. Finally, it automatically generates keys to help reading the applied visual variations.

7.2 Perspectives

The perspectives of this research work are numerous. We may think about enhancing the interactive keys for UML by integrating the other retinal variables, about the illustration of our guidelines via animated and interactive figures (in a website for

example), about studying animations and new visualizations in UML, about studying the use of the visual variables to facilitate collaboration around UML diagrams and obviously, about conducting the experiments using the defined design methodology. Future research can also concern the usability of diagrams in software engineering in a broader sense like helping practitioners to rapidly find accurate information in their diagrams or helping them to navigate between the views of their models. But, we choose to develop two other interesting perspectives of this work in the next two subsections: the operationalization of the SoG-UML guidelines in other new or existing Domain Specific Modeling Languages (DSMLs) and the enhancement of the interactive keys by integrating the crowdsourcing mechanism.

7.2.1 Operationalization of the SoG-UML guidelines in other DSMLs

To generate the SoG-UML guidelines, we followed a certain methodology. We begun by creating a feature model which describes the UML concrete syntax. Then, we treated the groups stemming from it to generate our guidelines, by avoiding to alter the UML primary notation. In that context, feature modeling is mainly used in the emerging software paradigm: the product line software engineering (PLSE). Feature models describe the common features (i.e., commonalities) and the variable ones (i.e., variability) of the products of a particular domain (e.g., cars in the car manufacturing domain). Their use aims at *developing highly reusable core assets for a product line* [32]. The same methodology (i.e., that we used in SoG-UML) might be automated for existing or new DSLs, where the products are guidelines of effectiveness of the retinal variables for these DSMLs and the domain is the software engineering modeling one. In fact, like UML, most of the DSMLs in this domain are composed of shapes which might contain text and might be contained in each other (i.e., commonalities). The starting point should then be a feature model which precises the visual variables of the primary notation and the graphic particularities of the language. This feature model can also be the same as defined for the UML concrete syntax, but, variability points should be defined to support the particularities of each DSML (for example, they can be extracted from a used profile which extends UML). Finally, the feature model can be parsed to prescribe a guideline for each node or variability point, based on the rules of the SoG.

7.2.2 Interactive keys and crowdsourcing

Crowdsourcing is a growing paradigm in which humans are the core computational entities. In [12], Brambilla et al. used the crowdsourcing to refine the graphical notation of DSMLs. This mechanism can be also used in our case to better take into account the preferences of the UML practitioners. They can make the choice of the implementation of a retinal variable that they deem effective and suitable to their contexts of use. Such mechanism can be integrated in a configuration page of the interactive keys. Each practitioner can rank the implementations and the most effective ones can then be defined. The mapping of the retinal variables to the information properties along with the effective categories can also be assessed via the crowd.

Bibliography

[1] Fliplayers eclipse wiki page. `https://wiki.eclipse.org/Papyrus/UserGuide/Layers`. accessed on 01-15-2019.

[2] Human factors in modeling workshop. `http://hufamo.compute.dtu.dk`. accessed on 13-05-2018.

[3] Lindholmen models repository. `http://oss.models-db.com/`. accessed on 01-27-2019.

[4] Object management group. `http://www.omg.org/`. accessed on 06-22-2018.

[5] Ongoing work around fliplayers. `https://wiki.eclipse.org/Papyrus/Oxygen_Work_Description/NewFeature/Layers`. accessed on 01-15-2019.

[6] Replication package of our empirical study about the visual variables in UML. `https://www.zenodo.org/record/827357#.XDNkNZNKiu4`. accessed on 01-07-2019.

[7] Color and daltonism. `http://daltonien.free.fr/daltonien/`, May 2003. accessed on 06-22-2018.

[8] Ahmar, Y. E., Gerard, S., Dumoulin, C., and Pallec, X. L. Enhancing the communication value of UML models with graphical layers. In *18th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MoDELS 2015, Ottawa, ON, Canada, September 30 - October 2, 2015* (2015), pp. 64–69.

[9] Andriyevska, O., Dragan, N., Simoes, B., and Maletic, J. I. Evaluating UML class diagram layout based on architectural importance. In *Visualizing Software for Understanding and Analysis, 2005. VISSOFT 2005. 3rd IEEE International Workshop on* (2005), IEEE, pp. 1–6.

[10] Bertin, J., Berg, W. J., and Wainer, H. *Semiology of graphics: diagrams, networks, maps*, vol. 1. University of Wisconsin press Madison, 1983.

[11] Blackwell, A., and Engelhardt, Y. A meta-taxonomy for diagram research. In *Diagrammatic representation and reasoning*. Springer, 2002, pp. 47–64.

[12] Brambilla, M., Cabot, J., Cánovas Izquierdo, J. L., and Mauri, A. Better call the crowd: using crowdsourcing to shape the notation of domain-specific languages. In *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering* (2017), ACM, pp. 129–138.

[13] Chaudron, M. R. V., Heijstek, W., and Nugroho, A. How effective is UML modeling? *Software & Systems Modeling 11*, 4 (2012), 571–580.

[14] Conversy, S. Unifying textual and visual: a theoretical account of the visual perception of programming languages. In *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software* (2014), ACM, pp. 201–212.

[15] Creswell, J. W. *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications, 2013.

[16] Dobing, B., and Parsons, J. How UML is used. *Commun. ACM 49*, 5 (May 2006), 109–113.

[17] Dykes, J., Wood, J., and Slingsby, A. Rethinking map legends with visualization. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 890–899.

[18] Dzidek, W. J., Arisholm, E., and Briand, L. C. A realistic empirical evaluation of the costs and benefits of UML in software maintenance. *Software Engineering, IEEE Transactions on 34*, 3 (2008), 407–432.

[19] El Ahmar, Y., Le Pallec, X., and Gérard, S. Empirical activity: Assessing the perceptual properties of the size visual variation in UML sequence diagram.

[20] El Ahmar, Y., Le Pallec, X., Gérard, S., and Ho-Quang, T. Visual variables in UML: a first empirical assessment. In *Human Factors in Modeling* (2017).

[21] Forward, A., Lethbridge, T. C., and Badreddin, O. Perceptions of Software Modeling: A Survey of Software Practitioners. Tech. rep., University of Ottawa, 2010.

[22] Garlandini, S., and Fabrikant, S. Evaluating the effectiveness and efficiency of visual variables for geographic information visualization. *Spatial information theory* (2009), 195–211.

[23] Genon, N., Perrouin, G., Pallec, X. L., and Heymans, P. Unlocking visual understanding: Towards effective keys for diagrams. In *Conceptual Modeling - 35th International Conference, ER 2016, Gifu, Japan, November 14-17, 2016, Proceedings* (2016), pp. 505–512.

[24] Gérard, S., Dumoulin, C., Tessier, P., and Selic, B. 19 papyrus: A UML2 tool for domain-specific language modeling. In *Model-Based Engineering of Embedded Real-Time Systems*. Springer, 2010, pp. 361–368.

[25] Green, T. R. G., and Petre, M. Usability analysis of visual programming environments: a cognitive dimensions framework. *Journal of Visual Languages & Computing 7*, 2 (1996), 131–174.

[26] Gregor, S. The nature of theory in information systems. *MIS quarterly* (2006), 611–642.

[27] Hebig, R., Ho-Quang, T., Robles, G., Fernandez, M., and Chaudron, M. R. V. The quest for open source projects that use UML: mining github. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems* (2016), ACM, pp. 173–183.

[28] IBM, I. B. M. C. The great mind challenge by ibm. `http://www-07.ibm.com/in/university/greatmind/tgmc_2009_bak.html`, 2009. accessed on 06-22-2018.

[29] Kosslyn, S. M. Graphics and human information processing: A review of five books. *Journal of the American Statistical Association 80*, 391 (1985), 499–512.

[30] Lanza, M., and Marinescu, R. *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Springer Science & Business Media, 2007.

[31] Larkin, J. H., and Simon, H. A. Why a diagram is (sometimes) worth ten thousand words. *Cognitive science 11*, 1 (1987), 65–100.

[32] Lee, K., Kang, K. C., and Lee, J. Concepts and guidelines of feature modeling for product line software engineering. In *International Conference on Software Reuse* (2002), Springer, pp. 62–77.

[33] LIST, C. Cascading stylesheets for UML in papyrus. `https://wiki.eclipse.org/MDT/Papyrus/UserGuide/CSS`, February 2013. accessed on 06-22-2018.

[34] Moody, D. L. The physicss of notations: toward a scientific basis for constructing visual notations in software engineering. *Software Engineering, IEEE Transactions on 35*, 6 (2009), 756–779.

[35] Object Management Group. Diagram definition. `http://www.omg.org/`, August 2015. accessed on 06-22-2018.

[36] PALLEC, X. L. Experiment about colors in UML. `http://www.lifl.fr/~lepallec/xpColor/`, February 2018. accessed on 06-22-2018.

[37] Purchase, H. C., Colpoys, L., Carrington, D., and McGill, M. UML class diagrams: an empirical study of comprehension. In *Software Visualization*. Springer, 2003, pp. 149–178.

[38] Recanati, C. Characteristics of diagrammatic reasoning. In *Proceedings of EuroCogSci07, the european cognitive science conference* (Delphi, Grèce, May 2007), D. K. edited by Stella Vosniadou and A. Protopapas, Eds., the second european cognitive science conference, Lawrence Erlbaum Associates, pp. pp 510–515.

[39] Riche, N. H., Lee, B., and Plaisant, C. Understanding interactive legends: a comparative evaluation with standard widgets. In *Computer graphics forum* (2010), vol. 29, Wiley Online Library, pp. 1193–1202.

[40] Sharif, B., and Maletic, J. I. An empirical study on the comprehension of stereotyped UML class diagram layouts. In *Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on* (2009), IEEE, pp. 268–272.

[41] Shull, F., Singer, J., and Sjøberg, D. I. *Guide to advanced empirical software engineering*. Springer, 2007.

[42] Störrle, H., and Fish, A. Towards an operationalization of the "physics of notations" for the analysis of visual languages. In *International Conference on Model Driven Engineering Languages and Systems* (2013), Springer, pp. 104–120.

[43] Termeer, M., Lange, C. F., Telea, A., and Chaudron, M. R. V. Visual exploration of combined architectural and metric information. In *Visualizing Software for Understanding and Analysis, 2005. VISSOFT 2005. 3rd IEEE International Workshop on* (2005), IEEE, pp. 1–6.

[44] Tudoreanu, M. E., and Kraemer, E. Legends as a device for interacting with visualizations. Tech. rep., Citeseer, 2001.

[45] Wong, K., and Sun, D. On evaluating the layout of UML diagrams for program comprehension. *Software Quality Journal 14*, 3 (2006), 233–259.

[46] Yusuf, S., Kagdi, H., and Maletic, J. I. Assessing the comprehension of UML class diagrams via eye tracking. In *15th IEEE International Conference on Program Comprehension (ICPC'07)* (2007), IEEE, pp. 113–122.

Scientific Publications by Yosser EL AHMAR

Conference Publications

[1] Ahmar, Y. E., Gerard, S., Dumoulin, C., and Pallec, X. L. Enhancing the communication value of UML models with graphical layers. In *18th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MoDELS 2015, Ottawa, ON, Canada, September 30 - October 2, 2015* (2015), pp. 64–69.

Workshop Publications

[1] Yosser El Ahmar, Xavier Le Pallec, and Sébastien Gérard. Empirical activity: Assessing the perceptual properties of the size visual variation in UML sequence diagram.

[2] Yosser El Ahmar, Xavier Le Pallec, Sébastien Gérard, and Truong Ho-Quang. Visual variables in UML: a first empirical assessment. In *Human Factors in Modeling*, 2017.

Appendix

Summary of SoG-UML guidelines

Following is a summary of the guidelines related to each retinal variable. Software practitioners can use them when they add significant visual decorations to their diagrams. For each retinal variable, we present its definition, perceptive properties and the methods to choose its effective categories. Then, we summarize its related guidelines in SoG-UML.

Color

The color variation is the differentiation generally induced by different colored excitations perceived between two ranges having the same brightness.
Human readers cannot understand the colors sensations if, first of all, the notion of color and the notion of brightness are not rigorously differentiated. Color and brightness are two different retinal variables. Each one is characterized by its perceptive properties.

1. PERCEPTIVE PROPERTIES

1.1 Perceptive attitudes

The color retinal variable is **associative and selective**: it can be used when a practitioner has an associative or a selective information property (e.g., distribution of tasks between the team members, the tested and non-tested classes).

It is **neither ordered nor quantitative**: it cannot be employed when a practitioner wants to convey an order or quantities which should be compared between each other (e.g., project progression, quality metrics)

1.2 Capacity

Table 1 mentions the capacity of the color variation: a practitioner should not exceed the number which is mentioned in the perceptive attitude of the color cell. The capacity of the color variation is the same for points, lines and zones. But, remember that UML nodes are considered as points and UML edges as lines.

Table 1: Perceptive properties of the color retinal variable.

	Associative	Selective	Ordered	Quantitative
Point	Unlimited	7	No	No
Line	Unlimited	7	No	No
Zone	Unlimited	7	No	No

1.3 Calculating effective categories of color

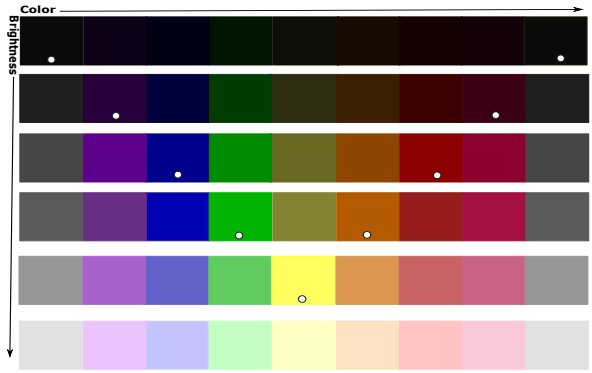

Figure 1: Colors of the spectrum in different values of brightness. Saturated colors are marked by a white circle.

- The selectivity is in its maximum near to the saturated colors and decreases by moving away (Figure 1)

- The choice of the selective colors is different depending on the brightness. Selective colors are near to saturated colors and it decreases by going away.
- **In big brightness (i.e., bright colors):** colors are chosen around the yellow, from green to orange. Light blue, violet, purple and red are less selective (Figure 1).
- **In medium brightness:** we find the maximum number of selective colors. The blue and red colors are diametrically opposite (Figure 1): violet, blue, green, yellow, orange, red and purple might be used in this level of brightness.
- **In low brightness (i.e., dark colors):** the selective categories of colors range from blue to red by the violet and purple. Dark green, yellow and orange are less differentiated (Figure 1).

If a practitioner wants to effectively chose his selective colors based on the previous method, he should use the Hue, Saturation, Brightness (HSB) system.
Light colors have a value of brightness (B) which is bigger than 70% (i.e., >= 150).
Medium levels of brightness are situated between 50% and 69% (i.e., 110<= B <150).
And, dark colors (i.e., low brightness) have a value of brightness which is lower than 50% (i.e., <110).

2. GUIDELINES ABOUT COLOR IN UML

Figure 2 shows the effective implementation of the color retinal variable in an UML class, Use Case, Decision and UML activity.

Figure 2: Effective implementations of the color retinal variable to 4 different UML nodes.

Following are the guidelines that we have defined in SoG-UML about color on UML node on a white background.

- [**SoG-UML #1**] Colors should not be applied to borders, lines of separation and/or text, when they are relatively thin.
- [**SoG-UML #2**] The thicker the border and separation lines are, the better colors are visually selected.
- [**SoG-UML #3**] Colors should be applied to the background of an UML node.
- [**SoG-UML #4**] Colors in medium levels of brightness should be used for UML nodes. They ensure the readability of black text labels, borders and separation lines and provide the biggest number of selective colors.

- [**SoG-UML #5**] In medium levels of brightness, text labels, borders and separation lines should be black.
- [**SoG-UML #6**] In high levels of brightness, text labels, borders and separation lines should be black.
- [**SoG-UML #7**] In low levels of brightness, text labels, borders and separation lines should be white.
- [**SoG-UML #8**] In medium brightness we find the maximum number of selective colors. The blue and red colors are diametrically opposite: violet, blue, green, yellow, orange, red and purple might be used in this level of brightness.
- [**SoG-UML #9**] In big brightness: colors are chosen around the yellow, from green to orange. Light blue, violet, purple and red are less selective.
- [**SoG-UML #10**] In low brightness (i.e., dark colors): the selective categories of colors range from blue to red by the violet and purple. Dark green, yellow and orange are less differentiated.

Guidelines about the effective implementation of color on icons:

- [**SoG-UML #11**] The border of icons should be black.
- [**SoG-UML #12**] Icons should not have the same background color as the UML node in which they are contained.
- [**SoG-UML #13**] All of the icons in a diagram should have the same background color.

Guidelines about the effective implementation of color on ports:

- [**SoG-UML #14**] Colors should be applied to the background of ports.
- [**SoG-UML #15**] The border of ports should be black.
- [**SoG-UML #16**] Ports should have a different selective color from the UML node in which it is embedded and the container of this latter.

2.1 Containment of UML nodes and color

Table 2 contains the guidelines about the containment colored UML nodes on different backgrounds.

Table 2: Containment relationship between UML nodes and grains.

Content/container	Content	Container
Color/Color	[**SoG-UML #37**] To get the color sensation, choose colors that are different from the container's color.	[**SoG-UML #38**] To ensure a better visual selectivity, the container should have a different color from the colors of its contained elements.
Color/Brightness	[**SoG-UML #39**] Choose colors having all the same brightness which is opposable to the brightness of the background.	[**SoG-UML #40**] The container should be as brighter as its contained elements are dark and vice versa.
Color/Grain	[**SoG-UML #41**] Increase the thickness of the borders and the lines of separation to better select the contained UML nodes in a possible very low value of grain.	[**SoG-UML #42**] Choose big values of grain for containers having more contained elements.
Color/Size	**No particular rules.**	**No particular rules.**
Color/Orientation	**No particular rules.**	**No particular rules.**

Brightness

Continuous progression that the eye perceives in the series of grays which ranges from black to white.
This progression is independent from the color and we can range from black to white by different levels of gray, blue, red, etc. Only one color should be used for such transitions.

1. PERCEPTIVE PROPERTIES

1.1 Perceptive attitudes

The brightness retinal variable is **selective and ordered**. When a practitioner needs to express a selective information property (e.g., elements of a pattern) or an ordered one (e.g., implementation progress), she/he can use it. However, it is **dissociative** because it inhibits the visualization of the UML nodes as a group.
The brightness is **not quantitative**, an UML practitioner should not use it to convey a quantitative information property.

1.2 Capacity

Table1 identifies the capacity of the brightens retinal variable in the perceptive attitudes which characterize it. An UML practitioner should not exceed the mentioned number in the corresponding cells. Note that the UML nodes are considered as points and edges as lines.

Table 1: Capacity of the brightness retinal variable

	Associative	Selective	Ordered	Quantitative
Point	No	3	Unlimited	No
Lines	No	4	Unlimited	No
Zones	No	5	Unlimited	No

1.3 Calculating effective categories of brightness

Categories of brightness have to be equidistant
For short number of categories, you can apply the following principles:
For 3 categories of brightness: choose black, medium gray (<50/50 (B/W)) and white, as shown in Figure 1.
For 4 categories of brightness, choose black, gray (25/75), gray (75/25) and white, as illustrated in Figure 2.

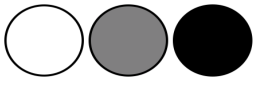

Figure 1: Three effective categories of brightness.

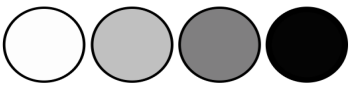

Figure 2: Four effective categories of brightness.

To generate a formula for finding equidistant categories of brightness, Jaques Bertin found that the only condition is to attribute a value different from zero to white and a value different from one hundred to black (which corresponds obviously to reality). In that context, if we denote:
W: The percentage of the White in the spot, W $\neq 0$.
B: The percentage of the Black in the spot, B $\neq 100$.
r: The common distance between the chosen categories of brightness. It denotes the ratio between the values of brightness of two successive categories (i.e., a logarithmic value). $r \in \mathbb{N}$.
n: Number of categories, $n \in \mathbb{N}$.

It is clear that we have to multiply the value of white W by the common difference r the number of wanted categories n minus one times. Therefore, Bertin defines the formula below:

$$B = Wr^{(n-1)}.$$

As a result, the common difference r will be:

$$r = \sqrt[n-1]{(B/W)}.$$

2. GUIDELINES ABOUT THE BRIGHTNESS IN UML

Figure 3 shows the effective implementation of the brightness retinal variable to an UML class, Decision, Use case and Activity. Below, we list the guidelines that we have generated in SoG-UML for the use of the brightness in UML.

Figure 3: Applying the brightness to the background is the effective implementation.

- **[SoG-UML #17]** The brightness variation should not be applied to the borders, separation lines and text: it is locked by the UML primary notation and not effective from the SoG point of view.
- **[SoG-UML #18]** The brightness should be applied to the background of UML nodes.
- **[SoG-UML #19]** Text labels, separation lines and borders should not have different values of brightness from each other.
- **[SoG-UML #20]** The brightness of icons is locked by the UML primary notation.
- **[SoG-UML #21]** The brightness of ports is locked by the UML primary notation.

2.1 Containment of UML nodes and the brightness retinal variable

Table 2 summarizes the guidelines about the containment of UML nodes having a brightness variation and all the possible retinal variables variation on their backgrounds.

Table 2: Containment relationship between UML nodes and brightness.

Content/container	Content	Container
Brightness/Color	[**SoG-UML #47**] Colors of the contained UML nodes should have a smaller level of brightness compared to their container.	[**SoG-UML #48**] The container of UML nodes which have a significant brightness variation should be white.
Brightness/ Brightness	[**SoG-UML #49**] The contained UML nodes should have a smaller value of brightness than their container.	[**SoG-UML #50**] The container should be brighter than its contained UML nodes.
Brightness/Grain	**No particular rules.**	[**SoG-UML #51**] The container should not have a grain variation or a very big value of grain where a big zoom is applied to the spots of the used texture.
Brightness/size	**No particular rules.**	**No particular rules.**
Brightness/ orientation	**No particular rules.**	**No particular rules.**

Size

The variation of the area of a spot represents the visual stimuli to the size variation.
Any spot, having a punctual or a linear signification, can vary its size without changing its position, brightness, grain, color, orientation or shape.

1. PERCEPTIVE PROPERTIES

1.1 Perceptive attitudes

- The size retinal variable allows all the perceptive attitudes except association. It is **selective, ordered and quantitative**, but, it inhibits the visualization of the UML nodes in one group because it is **dissociative**.
- The size retinal variable is the only quantitative retinal variable.

1.2 Capacity

Table 1 identifies the effective capacities of the size variation in the perceptive attitudes which characterize it.

Table 1: Capacity of the size retinal variable.

	Associative	Selective	Ordered	Quantitative
Point	No	4	Unlimited*	Unlimited*
Lines	No	4	Unlimited*	Unlimited*
Zones	No	5	Unlimited	Unlimited

Unlimited*: Unlimited, but the reader's eye cannot perceive more than 20 categories of size between two points having areas in the ratio of 1/10.

1.3 Calculating effective categories of size

The area of spots has to be proportional to the quantity of information property it expresses via the the variation of the radius of a circle, the side of a square or a triangle. The SoG provides an abacus of circles for all possible quantities.

2. GUIDELINES ABOUT THE SIZE RETINAL VARIABLE IN UML

Figure 1 shows the effective implementation of the size retinal variable to an UML class, Decision, Use case and activity if the corresponding diagram has a relatively small or medium visual density. Figure 2 shows the effective implementation of the size retinal variable in case the corresponding diagram has a relatively big visual density.

Following are the guidelines of SoG-UML about the use of the size retinal variable in UML.

- [**SoG-UML #22**] When the area of an UML nodes can be varied: the size retinal variable should be applied to the area of an UML node.
- [**SoG-UML #23**] The size retinal variable should be applied not only to the area of UML nodes, but also to the text, separation lines and borders.

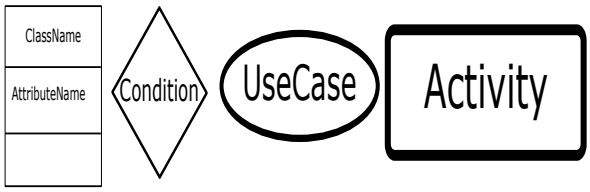

Figure 1: The effective implementation of the size retinal variable to a UML node.

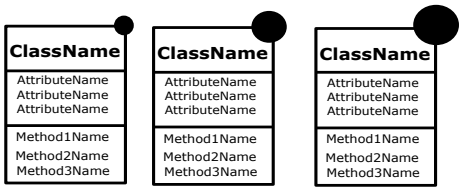

Figure 2: Effective implementation of the size retinal variable, in case the area of the corresponding UML node cannot vary.

- [**SoG-UML #24**] When the area of an UML node cannot vary: the size retinal variable should be applied to an additional point in the right corner of the UML node.
- [**SoG-UML #25**] To ensure a better selectivity, the icon's area should vary proportionally with the area of its related UML node.
- [**SoG-UML #26**] Varying the size of a port implies varying its area and its border lines thickness.

2.1 Containment of UML nodes and the size retinal variable

See Table 2.

Table 2: Containment relationship between UML nodes and size.

Size/Size	[**SoG-UML #43**] The size of the contained elements must be proportional to the size of their container. [**SoG-UML #44**] The biggest area of the contained elements is controlled by the size of its container. [**SoG-UML #45**] There is a limit under which the UML nodes are no more visible.	[**SoG-UML #46**] The smallest size of the container is controlled by its contained UML nodes.
Size/Color	**No particular rules.**	**No particular rules.**
Size/Brightness	**No particular rules.**	**No particular rules.**
Size/Grain	**No particular rules.**	**No particular rules.**
Size/Orientation	**No particular rules.**	**No particular rules.**

Grain

The grain is the succession of photo-graphical reductions of a semis of spots.
In a specific surface and for a regular semis, these reductions increase the number of spots, without varying the brightness. The grain is the quantity of separable spots contained in a unit surface. Broadly speaking, we can say that the grain corresponds to the magnification factor applied on the texture (if any). In a particular brightness, the grain is the quantity of separable spots contained in a unit area.

1. PERCEPTIVE PROPERTIES

1.1 Perceptive attitudes

The grain retinal variable is **associative, selective and ordered**.

1.2 Capacity

Table 1: Capacity of the grain retinal variable

	Associative	Selective	Ordered	Quantitative
Point	unlimited	2 to 3	unlimited	No
Lines	unlimited	3 to 4	unlimited	No
Zones	unlimited	4 to 5	unlimited	No

1.3 Calculating effective categories of the grain retinal variable

Categories of grain have to be equidistant.
The grain variation is associative. All its categories should have the same visibility (i.e., the same quantity of black per category). No particular rule is prescribed by the SoG.
Avoid using low values of grain: a lot of small spots
Low values of grain will be altered after printings. Graphic authors should prefer big values of grain: a small number of big spots.

2. GUIDELINES ABOUT THE USE OF THE GRAIN RETINAL VARIABLE IN UML

Figure 1 illustrates the effective implementation of the grain retinal variable in an UML class, Decision, Use case and Activity.

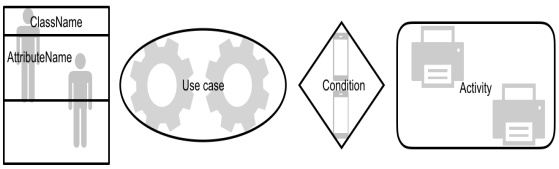

Figure 1: Effective implementation of the grain retinal variable.

- [**SoG-UML #27**] The grain retinal variable should be applied to the background of an UML node.
- [**SoG-UML #28**] Null values of grain where a lot of small spots are used should be avoided in UML.
- [**SoG-UML #29**] High levels of brightness should be applied to the used textures when applying grain variation.
- [**SoG-UML #30**] The background of icons should not have a grain variation.
- [**SoG-UML #31**] Satellites have not to change their grain. This variation is locked by the UML primary notation.

2.1 Containment of UML nodes and the size retinal variable

See Table 2.

Table 2: Containment relationship between UML nodes and grains.

Content/container	Content	Container
Grain/Grain	**No particular rules.**	[**SoG-UML #52**] The container should have a maximum value of grain.
Grain / Brightness	[**SoG-UML #53**] If the container is dark, then the grain variation has to be as brighter as possible.	[**SoG-UML #54**] The container should have a brightness as brighter as possible.
Grain/Size	**No particular rules.**	[**SoG-UML #55**] The size variation should not affect the grain variation of the contained nodes.
Grain/Color	**No particular rules.**	**No particular rules.**
Grain/Orientation	**No particular rules.**	**No particular rules.**

Orientation

A spot, in the form of a point, line or zone, can take an infinity of different orientations without changing its center.
We can feel the differences of orientations as far as the spot represents a linear aspect (i.e., the ratio Height/basis have to correspond to at least 4/1). The difference of angles between multiple parallel spots constitutes the stimuli of the orientation variation.

1. PERCEPTIVE PROPERTIES

1.1 Perceptive attitudes

- The orientation retinal variable is **associative**.
- The orientation retinal variable is **selective only in points (linear ones) and lines**.

1.2 Capacity

Table 1 summarizes the capacity of the orientation variation.

Table 1: Capacity of the orientation retinal variable

	Associative	Selective	ordered	Quantitative
Point	Unlimited	4	No	No
Lines	Unlimited	2	No	No
Zones	Unlimited	No	No	No

2. GUIDELINES ABOUT THE USE OF THE ORIENTATION RETINAL VARIABLE IN UML

Figure 1 illustrates the effective implementation of an UML node having a linear aspect.

Figure 1: Varying the orientation of UML nodes having a linear aspect.

- [**SoG-UML #32**] If the UML node has a linear aspect, the orientation of the whole UML node can be changed.
- [**SoG-UML #33**] The orientation retinal variable should not be applied to the background of UML nodes.
- [**SoG-UML #34**] Text, separation lines and borders should all vary their orientation for UML nodes having linear aspects.
- [**SoG-UML #35**] The orientation of icons should always have the same orientation of the main UML node.
- [**SoG-UML #36**] The orientation of the satellites should be the same as the UML node in which it is attached.

2.1 Containment of UML nodes and the orientation retinal variable

No particular rules for the containment of UML nodes having an orientation variation in all possibilities of background (i.e., containers presenting a retinal variable variation).

Abstract - UML is a visual modeling language for specifying, constructing, and documenting software intensive systems. UML diagrams play a central role in the whole software engineering process, starting from early analysis, through implementation, to maintenance. One of the main weaknesses of UML, pointed out by surveys of UML use in industry, is the difficulty to express the context of a particular diagram or to enrich it to enhance its communication value. In fact, UML does not specify effective ways to express contextual information (e.g., project management indicators, quality indicators). To express this kind of information in a particular visual language, the Cognitive Dimensions framework proposes to use its secondary notation: the visual variables that are not used by the language. Unfortunately, in UML, this secondary notation is not controlled. The first contribution of this thesis showed - via an empirical study - that this results in a big variety of usages of these visual variables and mostly, in not effective ways. It also showed that UML practitioners acknowledge the importance of the secondary notation in their practice while confirming their lack of knowledge about the associated effective practices. In this thesis, we refer to the Semiology of Graphics (SoG) in order to target an optimal use of the UML secondary notation. The framework SoG-UML is the result of our adaptation of the SoG to UML via 61 guidelines about the use of the visual variables in this language. We have realized a first experiment to validate a part of these guidelines. Finally, we propose Fliplayers and an interactive keys, two Papyrus plug-ins that respectively implement the layers mechanism and the construction of "effective" keys.

Keywords UML, Cognitive Effectiveness, Visual variables, UML secondary notation, Semiology of Graphics.

Résumé - Le standard UML est un langage visuel de modélisation pour spécifier, construire et documenter des systèmes logiciels. Une des faiblesses importantes d'UML identifiées par les études de la pratique réelle d'UML concerne la difficulté à spécifier le contexte pour un diagramme donnée ou à l'enrichir pour améliorer sa valeur communicationnelle. En effet, UML ne dispose pas de mécanismes efficaces pour exprimer ce genre de données (exp. les indicateurs de gestion de projets, les indicateurs de qualité). Pour les exprimer dans un langage visuel donné, le framework des dimensions cognitives propose d'utiliser sa notation secondaire : les variables visuelles que le langage n'emploie pas. Malheureusement pour UML, cette notation secondaire n'est pas régulée. La première contribution de la thèse a montré - à travers une étude empirique - que ceci résulte en une grande variété d'utilisation de ces variables et, dans de nombreux cas, de manière non efficace. Elle a aussi montré l'importance de la notation secondaire dans le quotidien des experts UML tout en confirmant leur manque de connaissance sur les bonnes pratiques associées. Dans cette thèse, nous nous basons sur la sémiologie graphique afin de viser une utilisation optimale de la notation secondaire d'UML. Le framework SoG-UML est le résultat de notre adaptation de la sémiologie à UML via 61 directives pour l'utilisation de variations visuelles dans ce langage. Nous avons réalisé une première expérimentation pour valider une partie de ces directives. Enfin, nous proposons Fliplayers et la légende interactive, deux plugins Papyrus proposant respectivement un mécanisme de calques et la construction de légendes « efficaces ».

Mots clés UML, Efficacité cognitive, Variables visuelles, Notation secondaire d'UML, Sémiologie graphique.