On using Colors in UML Models

Gefei Zhang
Hochschule für Technik und Wirtschaft Berlin, Berlin, Germany

Keywords: Modeling, Colors, Visual Aid, UML.

Abstract: Using colors has been recognized by Software Engineering research as helpful to make graphical software models more comprehensible. However, guidelines of where and how to use colors have been only little investigated yet. In this paper, we give two simple examples and propose some guidelines of where and how to use colors in UML models efficiently. Our guidelines may provide valuable help to use colors as a visual aid of software models and thus reduce the cognitive load of software developers in Model Driven Engineering.

1 INTRODUCTION

Color is an important visual element. It is said to "have the strongest effect on our emotions" (artyfactory, 2016). Colors are used in a large amount of graphical documents to show relationships between ordinal attributes, distinguish different categorical attributes, or to highlight certain parts of the document.

In Software Engineering, techniques of graphical modeling have been widely adopted. UML (OMG, 2015) and BPMN (OMG, 2011) are two prominent examples of successful graphical modeling languages. In graphical software models, however, colors still play an insignificant role; many models are monochrome, although using colors has been recognized as helpful to enhance the comprehension of software models, see (Reijers et al., 2011; Yusuf et al., 2007).

One of the reasons is that so far there has been only under proportional research on concrete guidelines of where and how to use colors in software models. In this paper, we show some examples of where colors may be used in software models, and, based on the examples, we discuss some dos and dont's of using colors. Our examples and discussion are based on the UML, while the principles are applicable to any graphical software modeling language.

The remainder of the paper is organised as follows: in the following Sect. 2 we show where to use colors by some examples of coloring model elements of UML diagrams, including both structural and behavioral models. Based on the examples, we discuss in Sect. 3 some dos and don'ts of using colors. In Sect. 4 we discuss related work, before we conclude and outline some future work in Sect. 5.

2 WHERE TO USE COLORS

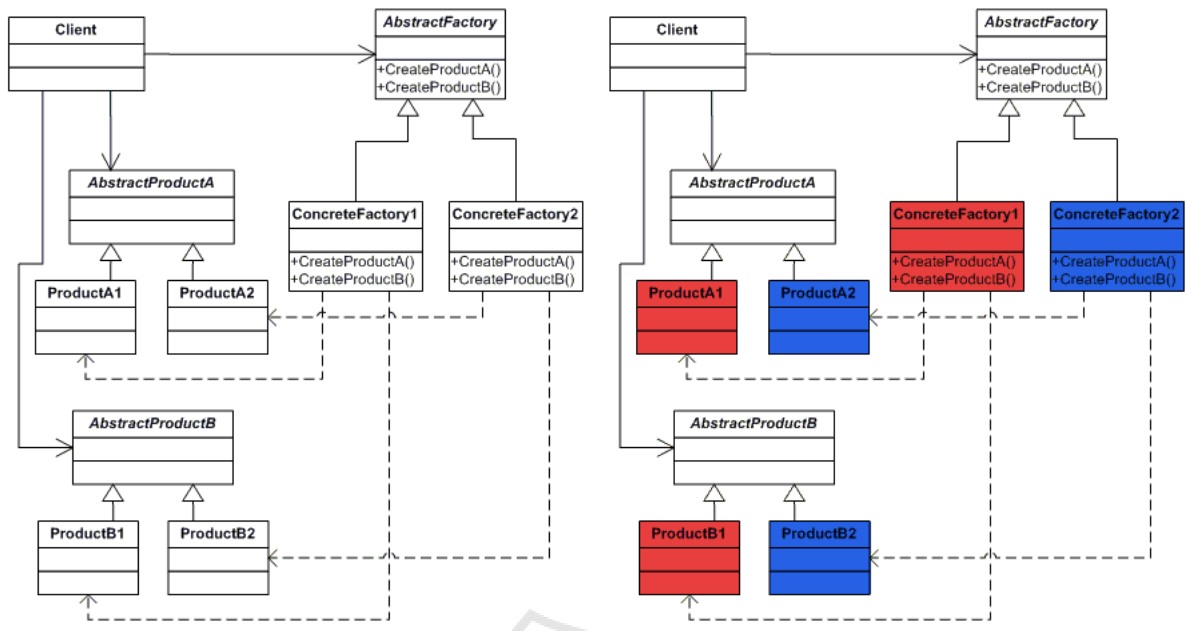
We show where colors may be applied in UML models by two examples. The first example is a class diagram modeling the structure of the well known *Abstract Factory* pattern (Gamma et al., 1994). In the second example we consider a state machine modeling a simple computer game.

2.1 Using Colors to Group Model Elements

In software models different colors may be used to put model elements into different groups. If model elements of the same group are colored with the same color, and model elements of different groups are colored with colors that are easily distinguishable from each other, then colors provide a visual aid for the reader of the model to grasp the groups and understand the model better.

For example, the class diagram shown in Fig. 1(a), which was taken from (doFactory, 2016), models the structure of the Abstract Factory design pattern, where ConcreteFactory1 is responsible to create instances of ProductA1 and ProductB1, while ConcreteFactory2 is responsible to create instances of ProductA2 and ProductB2. Since Fig. 1(a) is monochrome, the reader has to follow the arrows with dashed lines carefully, which represent UML dependencies relationships, to realize that

Zhang G.
On using Colors in UML Models.
DOI: 10.5220/0006262605090514
In *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2017)*, pages 509-514
ISBN: 978-989-758-210-3



(a) Black and white (b) Colored

Figure 1: Abstract Factory Pattern.

ConcreteFactory1, ProductA1 and ProductB1 belong to a product family, and that ConcreteFactory2, ProductA2 and ProductB2 belong to another product family. If there were more product families, this relationship would be even less obvious.

In Fig. 1(b), we use colors to emphasize the relationship of "belonging to the same product family": ConcreteFactory1, ProductA1 and ProductB1 are colored red, and ConcreteFactory2, ProductA2 and ProductB2 are colored blue. The classes of the same color are easily perceived as having "something in common", while the classes of different colors clearly belong to different groups. The product family relations are therefore made more apparent.

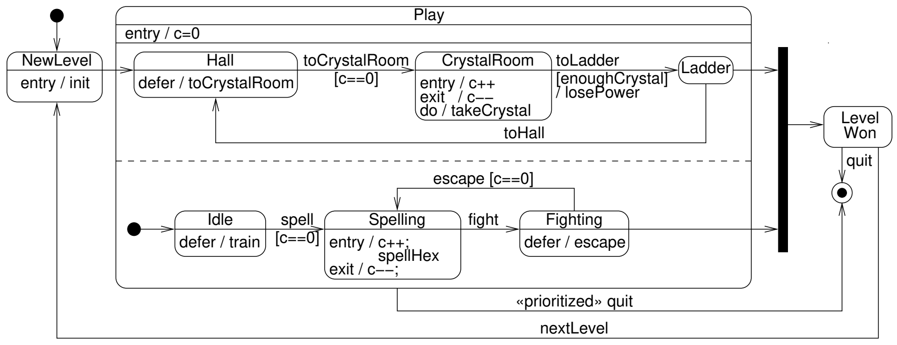
2.2 Using Colors to Show Special Behaviors

In behavior models, relationships between model elements are usually even more complex. The model reader often needs to understand the interactions of a whole number of model elements in order to understand the relations between the model elements. Using colors to highlight these relationships may provide a visual aid to understand the model more easily.

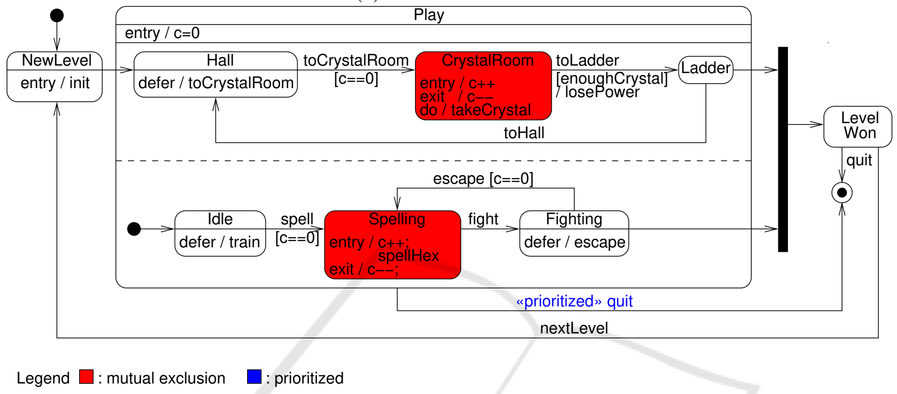
For example, Fig. 2(a), which was taken from (Zhang, 2012), shows a UML state machine modeling the behavior of a player during a certain part of a computer game. Very simply speaking, the player—a magician—starts in a state where they has to chose a NewLevel. Upon completion of the preparations they is transferred into the Play state which contains two concurrent regions, modeling two different aspects of the magician's intelligence. The upper region describes the possible movements of the player: in each level they initially starts in an entrance hall (Hall), from there they can move to a room in which magic crystals are stored (CrystalRoom) and on to a room containing a Ladder. From this room the player can either move back to the hall or, after fighting with some computer figure and winning, exit the level.

The lower region specifies the magician's possible behaviors. They may be Idle, gathering power for the next fight, Spelling a hex, or Fighting. They may escape from the fight and try to spell another hex, or, if they wins the fight in the Ladder room, win the level and move on to another level. Any time while Playing, they can leave the game and quit. Note that the event quit has a stereotype ≪prioritized≫, meaning that it has a high priority and its transition should be fired at once when quit is received, even if there may be other events in the event pool.

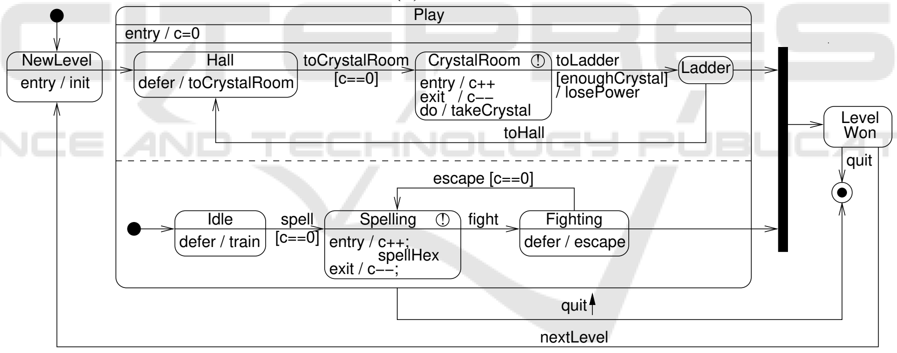
Moreover, the magician modeled in Fig. 2(a) is not allowed to spell a hex in the crystal room. That is, the states CrystalRoom and Spelling must not be simultaneously active. This mutual-exclusion rule is modeled by the interaction of a bunch of model elements: A variable c is introduced and used to control the access to the two critical states: it is initialized as 0 in the entry action of Play, increased whenever CrystalRoom or Spelling is activated, and decreased whenever one of



(a) Black and white



(b) Colored



(c) Using icons

Figure 2: State machine with mutual exclusion.

the two states is deactivated. The three transitions that activate the two states (from Hall to CrystalRoom, from Idle to Spelling, and from Fighting to Spelling) are guarded by conditions, such that they are only fired when c equals 0, which means that the other critical state is currently inactive and the mutual exclusion rule is satisfied. A subtle point is that we have to declare the events toCrystalRoom, spell, and escape to be deferrable in the states Hall, Idle, and Fighting respectively. In this way the transitions are postponed if the other critical state is active, and will be automatically resumed without requiring the events to be sent again. Otherwise the events would be lost in case exactly one of the critical states were active, since the event would then be taken from the event pool without firing a transition.

Since the mutual exclusion behavior is modeled by the interaction of a bunch of model elements, including increment and decrement of variables, checking guard conditions at several transitions, and deferring events, the behavior is hard to comprehend.[1] However, we can fill the states CrystalRoom and Spelling with the same color (in Fig. 2(b): red) to highlight that these states have "something special", and, since they share the same color, this something special is actually something in common. Moreover, we also give

[1]Actually, this is an intrinsic issue of standard UML state machines, see (Zhang, 2012).

the event quit a color (in Fig. 2(b): blue), highlighting that it has a special semantics (being handled immediately, even if there are also other events in the event pool).

Additionally, we put a legend in Fig. 2(b) to make clear that in the figure, the color red means mutual exclusion, and that the color blue means high priority. This way, the fact that the two states are mutually excluded is efficiently alerted to the reader of the model.

3 DOS AND DON'TS

Based on the above examples of where to use colors appropriately in UML class diagrams and state machine models, we discuss some dos and dont's of color usage, and give a brief comparison of using colors and icons that may be defined as part of stereotypes, one of the standard UML extension mechanisms.

Use Colors only to provide Redundant Information, don't rely Solely on Colors

Colors, as compared with shapes or texts, cannot always be seen properly. For example, modelers and software developers may be color blind, or a software artefact may be printed on paper by a monochrome printer. Therefore, it is very important not to present information by colors only. Colors should only be used to *highlight* some information that are already modeled otherwise. The purpose of using colors in graphical models is rather to *enhance* the user experience, letting the user to understand the model more easily, than to provide information modeled by colors only.

Keep Colors consistent with the Model, avoid Inconsistency

Since the information provided by colors should be redundant, it is important to make sure that this information is kept consistent to the model when the model is modified. For instance, if our computer game evolves to a new version and the behavior of the computer player in Fig. 2(b) is changed such that the event quit is no longer prioritized, then its color should also be removed. Otherwise the reader would only feel confused.[2]

[2]This phenomena is well known in programs with comments that no longer match the code.

Use colors only to provide qualitative information, don't model precise quantitative information by colors.

The human cognition of colors is imprecise. Color nuances are hardly perceived. Therefore, it is better to use different (distinguishable) colors for model elements in different categories, rather than using similar colors for quantitative differences.

Use Colors only to provide Simple, Highlighting Information, don't Model Complex Logic with Colors

Colors are a very good means for highlighting and alerting. Usually, colors are good to represent simple notifications rather than complex logic. For instance, in Fig. 2(b), we used colors only to show that the two states have something in common, and it is the legend that explains what the common thing is. More detailed logic, such as how the state machine waits to enter the critical states, should not be modeled by colors.

Consider using a Legend, don't let the Reader Guess

If some special semantics (e.g., mutual exclusion in Fig. 2(b)) is highlighted by a color, then it may be helpful to provide a legend, which explains very briefly what the color means in a few words. Otherwise the reader would only see that the colored model elements are "somehow" different, but usually do not understand what the difference is.

Prefer Using Colors to Icons

Icons are also used in the UML to show that some elements are "somehow different". Usually, these elements are stereotyped, and defined in a so called UML *profile*. Stereotyping is actually one of the standard extension mechanisms of the UML. A stereotyped model element usually has a semantics which is related to, but different than that of the base element, and may be indicated by either the name of the stereotype, enclosed in guillemets ≪ and ≫, or by an icon, specified by the stereotype's designer. For instance, in Fig. 2(a), the event quit has a stereotype ≪prioritized≫, meaning that it has a high priority and its transition should be fired at once when the event is received, even if there may be other events in the event pool.

In Fig. 2(c), we use icon (!) to indicate that the states CrystalRoom and Spelling are stereotyped, and the icon ↑ to indicate that the transition from Play to the final state is prioritized. Note that usually the name and actual semantics of the stereotype are defined in another document, a so called UML *profile*. However, the icons do alert to the reader of the model that the two states and the transition are different than normal, non-stereotyped states and transitions.

While icons can also be used to show that some model elements are different than standard model elements, they usually take only a small part of the model element, and are less prominent than colors. It is easier to see colors than icons. Also, difference between colors is easier to perceive than difference between icons. Therefore, colors are more suited to highlight differences between model elements. On the other hand, colors and icons may be used complementarily, and icons may be colored. The above discussion about dos and dont's also applies to using colors in icons.

4 RELATED WORK

It has been recognized that colors may be helpful to enhance the understanding of software models, see (Yusuf et al., 2007), focusing on UML class diagrams, and (Reijers et al., 2011), focusing on Petri Nets, although these papers do not contain guideline for the use of colors. Some guidelines regarding layout of model elements are published in (AgileModeling, 2016), and in (Mendling et al., 2010) the authors make recommendations of the elements to use in graphical models, e.g., that OR branching should be used as little as possible. In these guidelines, colors are not considered. To the author's knowledge, our paper is novel in that we provide guidelines of where and how to use colors in UML class diagrams and state machine models.

In programming, modern IDEs, such as Eclipse[3] and IntelliJ IDEA[4], usually display different elements of a program, such as keywords, identifiers, numbers, strings, comments, etc., in different colors, and users can have individual settings of these colors. Moreover, simple static analysis techniques are applied, so that erroneous code and code with bad smells, e.g., unused variables, are also shown in different colors than "normal" code. This way, the programmer receives visual aids to structure and understand the program better, and to see potential errors in the program. To the author's knowledge, guidelines of how to use colors in programs have not been investigated and published yet.

[3]http://www.eclipse.org/

[4]http://www.jetbrains.com/idea/

5 CONCLUSIONS AND FUTURE WORK

We have given two examples of where to use colors as a visual aid in UML diagrams. We considered both structural and behavioral modeling. Based on the examples, we proposed some guidelines of how to use colors in UML diagrams.

Future work includes development of modeling tools to enable convenient modeling with colors, and empirical studies on the effectiveness of our guidelines. We also plan to extend our guidelines to cover other graphical modeling languages, such as BPMN and Petri Nets.

ACKNOWLEDGEMENTS

This work has been partially sponsored by the project "Anwendungszentrum creative Applied Interactive Technologies" (cAPITs) of the European Regional Development Fund.

REFERENCES

AgileModeling (2016). Modeling Style Guidelines. http://agilemodeling.com/style/. Accessed on 2016-11-16.

artyfactory (2016). The Visual Elements—Color. http://www.artyfactory.com/art_appreciation/visual-elements/color.html. Accessed on 2016-11-16.

doFactory (2016). http://www.dofactory.com/net/abstract-factory-design-pattern. Accessed on 2016-11-16.

Gamma, E., Johnson, R. H. R., and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison.

Mendling, J., Reijers, H. A., and van der Aalst, W. (2010). Seven Process Modeling Guidelines (7PMG). *Information and Software Technology*, 52(2):127–136.

OMG (2011). Business Process Model and Notation, Version 2.0. Specification, Object Management Group. http://www.omg.org/spec/BPMN/2.0/PDF/, Accessed on 2016-11-16.

OMG (2015). Unified Modeling Language, Version 2.5. Specification, Object Management Group. http://www.omg.org/spec/UML/2.5/PDF/, Accessed on 2016-11-16.

Reijers, H. A., Freytag, T., Mendling, J., and Eckleder, A. (2011). Syntax Highlighting in Business Process Models. *Decision Support Systems*, 51(3):339–349.

Yusuf, S., Kagdi, H., and Maletic, J. I. (2007). Assessing the Comprehension of UML Class Diagrams via Eye Tracking. In *Proc. 15th IEEE Int. Conf. Program Comprehension (ICPC'07)*, pages 113–122. IEEE.

Zhang, G. (2012). Aspect-Oriented Modeling of Mutual Exclusion in UML State Machines. In Vallecillo, A., Tolvanen, J.-P., Kindler, E., Störrle, H., and Kolovos, D. S., editors, *Proc. 8^{th} Eur. Conf. Modeling Foundations and Applications (ECMFA'12)*, volume 7349 of *Lect. Notes Comp. Sci.*, pages 162–177. Springer-Verlag.