Colored-Object Programming : about the Ergonomy of the Visual Formalism

Henry J. Borron

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Colored-Object Programming : about the Ergonomy of the Visual Formalism

Henry J. Borron

N° 2879
Avril 1996

THÈME 2

INRIA
SOPHIA ANTIPOLIS

Colored-Object Programming : about the ergonomy of the visual formalism.

Henry J. Borron *

Programme 2 — Génie Logiciel et Calcul Symbolique
Action LeTool

Rapport de Recherche N° 2879 — Avril 1996 — 26 pages

Abstract. This paper discusses the ergonomy of the visual formalism proposed for a refinement of object oriented programming termed "colored object programming".

A number of principles are first proposed (ex. : unique source, unique destination, non ubiquity,...). With the help of examples, these principles are shown to mould both the color graph formalism and its specific instantiations. These principles may be transgressed to yield new propositions : corresponding advantages and drawbacks are discussed.

Two important alternatives are then examined : cartesian type representations useful from a pedagogical point of view, yet obviously unmanageable in general ; and what we call "color higraphs", obtained by using insideness in place of connectedness for reflex transitions. Except for a few details, this transformation remarkably leads to the David Harel's notation (i.e. higraphs or more exactly, statecharts) for -a priori, non object-oriented- state-transition systems.

To conclude, we propose to center the planned programming environment around our visual formalism, and to open it to alternative representations depending on the programmer's task (as a writer or a reader).

Keywords. Cognitive ergonomy, cognitive dimensions, visual formalism, language design, color graphs, higraphs, object oriented programming, state, transition.

(Résumé : tsvp)

* borron@chris.inria.fr

INRIA, Unité de Recherche de Sophia-Antipolis
2004, route des Lucioles. BP 93, 06902 SOPHIA-ANTIPOLIS Cedex (France)
Téléphone : (33) 93 65 77 77 — Télécopie : (33) 93 65 77 65

Colored-Object Programming: About the visual formalism.

1. INTRODUCTION

Colored Object Programming (COP) is a promising offspring of Object-Oriented Programming (OOP). Say in a very short manner, COP is an effort to push to its ultimate point the basic idea of OOP, i.e. to give data-structures the responsability of operations they can support. In COP, the behaviour of an object depends not only on its class but also on its state.

COP promotes states and transitions as fundamental features, and relegates memory representations and methods as second citizen features (implementation level). Using states and transitions, COP enables program designers to express as a whole a class interface (more exactly, the programming interface of the instances of a class). COP supports inheritance of transitions, inheritance of memory representations and inheritance of methods both in a color graph and -by generalization- in a class hierarchy.

The color graph visual formalism is described in [Borron, 1996c] and, in a more accessible form, in [Borron, 1996a]. We expect it to be easily understood by humans (because it is visual[1]) as well as by computers (because of its precise semantics [Borron, 1996c] [Borron, 1996d] [Borron, 1996e]). Although we expect it to be mature and fairly intuitive, we are opened to improvements or even to moving to a different notation if this one appears to be superior from a cognitive point of view.

Details may have important effects. In matter of languages, a *"trivial change could measurably affect performance"* and *"language designers need to be extremely careful about even the tiniest details"* ([Fitter-Green, 1979], p.252). This warning was made after observing users' difficulties in selecting a sequence of statements for joining a source to a destination : the problem *"was considerably more difficult"* when statements were expressed in the form *"You can get to B from A"* instead of *"From A you can get to B"*.

[Green, 1982] also expresses the same warning in several places[2], notably in the conclusion (p. 34) : "*The details of the notation profoundly influence its usability, much more so than would at first suspect (...) The effect of detail is made less surprising when one considers the explosive effects of scaling up a simple algorithm (...) to something ten, a hundred, or even a thousand times bigger*". Hence, "*A saving too small to be measured [on a small scale] could become a major improvement a thousand times up.*"

This paper presents the arguments justifying the current definition of color graphs : first, we discuss their design and use on the basis of a few principles ; then, we relate color graphs with alternative visual representations. A cartesian type of representations appears interesting from a theoretical point of view (a color graph is a representation of instance states in a N-dimensions space) ; the comparison between higraphs, a formalism proposed by David Harel for reactive systems, and color graphs contributes to the debate between insideness and connectedness.

2. COLORED OBJECT PROGRAMMING : AN EXAMPLE

Let's very shortly describe here the semantics of our formalism and illustrate it –briefly, too– with a fairly simple example.

2.1 Color graphs : an overview

2.1.1 Semantics

COP is a refinement of OOP : the answer of a colored object to a "message"[3] depends not only on its class, but also on its current state. The term "color" is used in a metaphorical way (idioms often translate a state as a color ; for example, someone may be green with envy, red with shame, etc.).

---

[1] The visual formalism has a textual counterpart. May this part easier to read and understand for certain programmers ? We do not think so. An interesting reference is [Scanian, 1989].

[2] Refer to the same paper, pp. 21-22, for an analysis of fall-backs in some type of flowcharts.

[3] COP supports multiple dispatch (selection of methods according to several arguments as done in CLOS). For ease of reading, we use the term "message" which refers to single dispatch (selection of methods according to one argument as done in Smalltalk).

A class is abstracted as a whole in the form of a **color graph** (the class interface), describing **states** and **transitions** between them. States are described according to either one or several **dimensions** (this is the central point) : in the first case, each state is represented by a single node (a **color**) : the color graph is termed a **c-graph** ; in the second case, each state is basically represented by N nodes, one per dimension (each such node is termed a **pigment**) : the color graph is termed a **p-graph.** Each node in a color graph is defined by a **condition** (evaluated by sending a side-effect free message to the instance) and/or results from a **construct** (see below). Transitions come in two kinds : **regular transitions** that fire on an external event (a "message") ; **reflex transitions** that fire on internal conditions (more precisely, a reflex transition fires when the condition attached to its destination node gets true). On top of reflex transitions are built constructs (**selection, factorization**[4], **decomposition, conjunction**), either diverging (the first three) or converging (the last one), their root node being either of type OR (first two constructs) or AND (last two constructs). Regular transitions are **inherited** along reflex transitions (of all constructs except the decomposition). A reflex transition may thus have two possible roles : one dynamic (firing, at run time) ; one static (factorization, before execution). Given an instance, its current state is **marked** in a c-graph by a single **token** and in a p-graph by several **mini-tokens**, one per dimension. When firing, a transition moves the mark (token or mini-token(s)) from its source node(s) to its destination node(s) ; then, depending on the new instance state, one or more reflex transitions may fire in turn, hence determining the new position of the mark.

A color graph may be given one or several implementations. For each one, a **memory representation** may be attached to each node ; a **pre-** and a **post-method**, to each regular transition. Exactly like regular transitions, implementation items (representations and **micro-methods**) are **inherited** along the reflex transitions of a given color graph. The item to be used in one node results from a **combination** of the items inherited in this node (one item per involved dimension). **Class inheritance** is defined as a generalization of the local inheritance and combination rules. Smalltalk simple inheritance and combination mechanism may be mimicked (without the non modular "super" construct) ; CLOS' multiple inheritance and sophisticated combination mechanisms, too (without the non modular "call-next-method" construct). Pre-methods are executed first (from the most specific to the least specific) ; then post-methods (from the least specific to the most specific). Execution of all micro-methods is systematic unless a couple is declared as *:masking* : such a declaration prevents less specific couples of micro-methods to be run. Hence, the combination order is purely declarative (it can be predicted by the programmer solely by looking at the method headers).

2.1.2 Visual notation

A textual notation (derived from CLOS[5]) and a visual notation have been elaborated. The visual one is illustrated just below by the example. Let's give some preliminary indications about it.

A node is pictured as a small bubble : inside the bubble, a name or an integer identifies the node ; close to the bubble, the condition attached to this node is named (inferred conditions are usually not displayed). Regular graph transitions are represented with named plain arrows ; reflex transitions, by unnamed dashed arrows (using a dotted line for selections, a broken line for conjunctions). A decomposition is not represented using dashed arrows, but by a small bar between the root node and the leaf nodes (the root node is usually not shown in fact). One unnamed arrow usually ondulates from a special node to an initial color (numbered 1 by default). This corresponds to the default **creation transition** (*make-instance*)[6]. The special node corresponds to the case an instance is not created or has been destroyed : noted _ (underscore) in textual syntax, it is represented as a small segment with the class name above it in the visual interface. **Testing transitions** (i.e. regular transitions associated to side-effect free messages used for evaluating conditions) are automatically derived by the system and are thus usually not shown.

2.2 A color graph example

Next figure is derived from the source code of an example given in section 4.2 of reference [Chambers, 1993][7]. *Person* instances are differentiated into *male* and *female* categories according to the *sex* ; into *child, teenager* and *adult* categories, according to the *age*. In addition, both criteria are used to distinguish *boy* and *girl* categories.

In terms of COP, *sex* and *age* form two **dimensions**. The *Person* color graph has thus a **degree** 2. It exhibits a **decomposition** of **color** 1 (representing the uninitialized instance state) in two **pigments** (2 and 6). Pigment 2 pertains to the *sex* **scale** ; in the same scale, are pigments 3, 4 and 5. Pigment 6 pertains to the *age* scale ; in the same scale, are pigments 8 to 12. Pigments 3 is an **ephemere** pigment : it is the root of the **selection** on pigments 4 and 5, which are both **basic** pigments. In the *age* scale, nodes 10, 11 and 12 are basic pigments ; and nodes 7, 8 and 9 are ephemere ones. **Nodes** 13 and 14 result from a **conjunction** : they are not pigments but **blends** of **degree** 2 (because they group

---

4 The factorization construct is not used in practice ; it is mentioned here for explaining our design (see subsections 4.2.2 and 3.1).

5 For method names, we use the Smalltalk syntax. This one states the number of expected arguments. Three types of method names (also termed **selectors**) exist : (1) unary selectors (like *factorial*) : no argument expected ; (2) binary selectors (like +, -, *, ...) : one argument is expected ; (3) keyword selectors (like *insert:at:*) : the number of colons state the number of expected arguments (example of message : *'foo' insert: 'bar' at: 2*).

6 Several creation transitions may exist : in such a case, they should be named (the default one is cancelled).

7 For the purpose of the demonstration, a slight difference exists to create an instance of *Person* : the cited paper proposes a *make-person* method with values for *sex:* and *age:*. The corresponding creation transition is not represented in figure 1.

two pigments) ; they may also be termed colors –in this example– since their degree is equal to the color graph degree (they group a pigment of each scale). (A pigment has a degree 1.) Nodes in a p-graph (pigments, blends) are collectively termed **p-chromas**. To each **chroma** (color, p-chroma) of a color graph is associated a **condition**, which may be user-defined (for example, *child* for node 10) and/or system inferred (for example, *child* OR *teenager* for node 8). The condition of node 13 is *male* AND *child* (*male.child*, for short) and is also renamed as *boy* by the user. Besides chromas and trees of **void transitions** (due to selections and conjunctions), the color graph exhibits **regular transitions**, telling what messages are **acceptable** given an instance state and how they possibly modify it. Transition *age:*, for example, is acceptable in states (*, 6) where * may be any pigment in the *sex* scale. Once this transition has occurred, the **new state** is (=, 7) where = means that the first pigment remains unchanged. *Expected-lifespan* is **factorized** : the corresponding message can be sent while the instance is in 4 and 5 (and thus in 13 and 14). The same message is not acceptable in 3 (only testing messages are acceptable in ephemere pigments ; they are sent automatically). *Bedtime* is also factorized. *Long-lived*, which tells if an instance is older than its *expected-lifespan*, is a **constrained transition** : it is acceptable only when the instance state derives from (3, 7) : states 13 and (4, 11) are such cases. Node 3 is the source state of the transition ; node 7 is specified by the **clause** "[7]" that follows the name of the transition. Note the *have-birthday* transition cannot be factorized in 7 without loosing in precision concerning its destinations. Except for *sex:* and *age:*, all mentionned transitions are inherited in 13 and 14. Finally, because the testing transitions (ex. : sex, *child*, *boy*) are automatically built by the underlying system, they are not represented. Given an instance, two **mini-tokens** —one per dimension— mark the current state : in the figure below, the instance is supposed to be in state (2, 10).

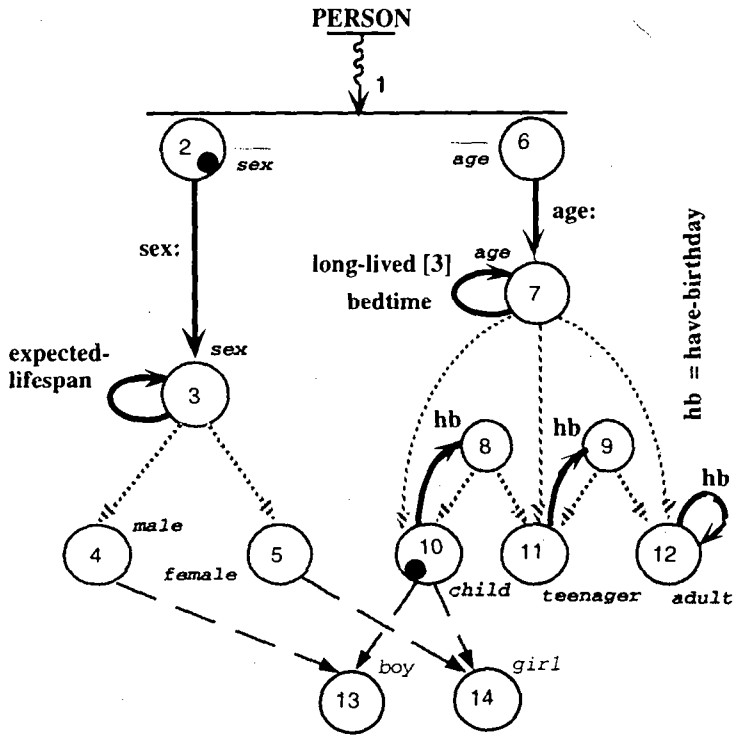


Figure 1.

PART I : DESIGN PRINCIPLES

In this first part, we describe the design of the notation used for visually expressing the COP formalism, as well as principles for choosing between alternatives when instantiating a color graph. When appropriate, justifications are even brought at the formalism design level itself. Role expressiveness, an important cognitive dimension [Green, 1989], was often a reason motivating our choices and principles. It is first shown as underlying a notable part of the visual notation. Five practical principles, sometimes deriving from role expressiveness, are then formulated for complementing the design of the visual notation and/or guiding the design of a color graph instance.

3. A HIGH-LEVEL PRINCIPLE : ROLE EXPRESSIVENESS

As underlined just above, role expressiveness largely motivates the design of the COP visual notation.

3.1 color graph specifications

— First of all, the decision itself to visually represent the formalism, i.e. chromas and transitions, was itself motivated by a serious concern for expressiveness : a textual representation is possible (it exists), but it is by far a less expressive one. Note also that our choice in favor of connectedness (reflex transitions) is also partly motivated by expressiveness.

— The respective roles of regular transitions and reflex transitions are different : the formers model the possible occurrences and the effects of external events ; the latters, the internal evolution of an instance to a stable or meta-stable position once freed from the impulse of an external event[8]. Being different, these roles need to be distinguished at the notation level : the formers are visually represented with named plain arrows ; the latters, by unnamed dashed arrows.

— What is less obvious at first sight is the necessity of visually distinguishing the reflex transitions of a selection from those of a conjunction. Yet, the next two figures, extracted from figure 1, show the possible existence of pseudo-selections and pseudo-conjunctions. A **pseudo-selection** looks like a selection, but the reflex transitions it is made of participate in fact to different conjunctions flowing from a same p-chroma. A **pseudo-conjunction** looks like a conjunction, but the reflex transitions it is made of participate in fact to different selections flowing to the same chroma. If no visual distinction is made between the reflex transitions of a selection and the relex transitions of a conjunction, then the user possibly faces visually ambiguous structures. Of course, given the semantics of each node, the user is able to disambiguate the drawing, but at the expense of some mental overhead. The problem is probably more severe to the reader of a color graph than to his/her writer : the latter has in mind the structure and he/she can even not notice the visual ambiguities (if any) were no distinction been made ; the former, because he/she needs to build up a mental structure from the drawing, is more sensitive to these details. Thus, for role expressiveness, a distinction was made between the two types of reflex transitions : a dotted line is used for those of a selection ; a broken line for those of a conjunction.

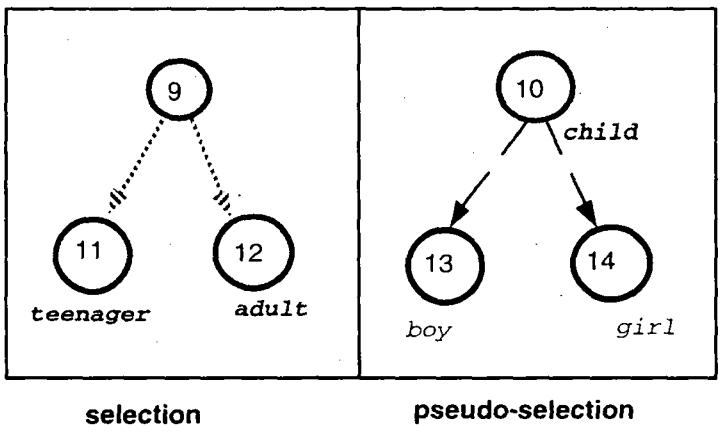


Figure 2.

[8] A useful metaphor developed in [Borron, 1996a] when comparing a color graph to a pinball. In a c-graph, one ball exist (the token). Creating an instance is like launching the ball ; an external event is like acting on a flipper : it moves the ball to a new position depending on the characteristics of this event (energy, direction) ; once acted on, the ball rolls by itself until it founds a stable or meta-stable position. The reflex transitions modelizes these latter moves. In a p-graph, there are more balls (one per mini-token, i.e. dimension), but the metaphor still holds.

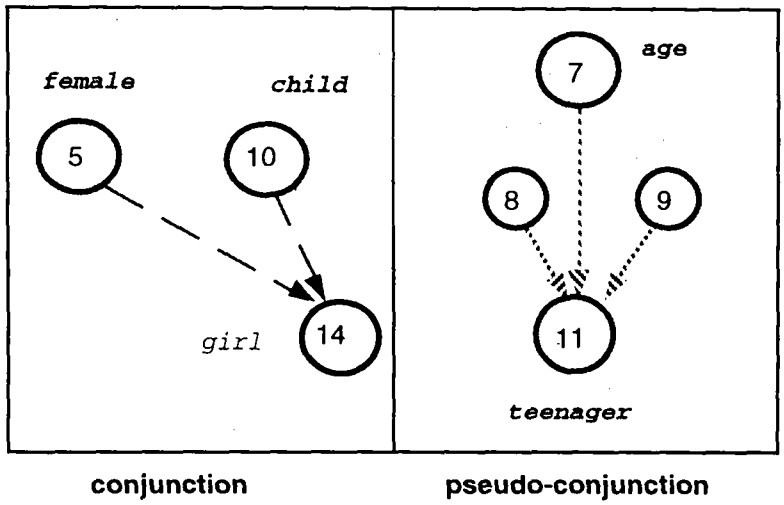


Figure 3.

— Another problem is posed by the representation of a decomposition —and, accessorily of a factorization. The semantics of a decomposition is formally defined in terms of the reflex transitions it is made of. As a matter of fact, this statement is also true vs. selections and conjunctions, as well as vs. factorizations. The concept of reflex transition is simple (a reflex transition fires when the condition attached to its destination node gets true) and all four constructs derive from it. From a design point of view, this hierarchical structuring is quite satisfying as far as the concepts are concerned. Yet, in practice, the respective topologies of a selection, of a decomposition, and of a factorization are the same : all three are diverging constructs. Visually, this similarity is a potential source of confusion, even if the semantics is quite different (a decomposition is a AND construct ; other diverging constructs are OR constructs). We settle on the problem (1) by ruling out the factorization construct (cf. subsection 4.2.2) ; (2) by providing a quite different representation for the decomposition.

To avoid any confusion with a selection, a decomposition is not graphically represented using unamed dashed arrows (i.e. reflex transitions), but by a small bar between the color node (usually not shown) and the pigment nodes. This representation accurately represents the underlying AND construct. This representation presents an other advantage : the local inheritance rule is easily visualized using this representation since the reflex transitions of the decomposition –along which regular transitions cannot precisely be inherited (sole exception)– are not pictured. Our choice thus favors consistency.

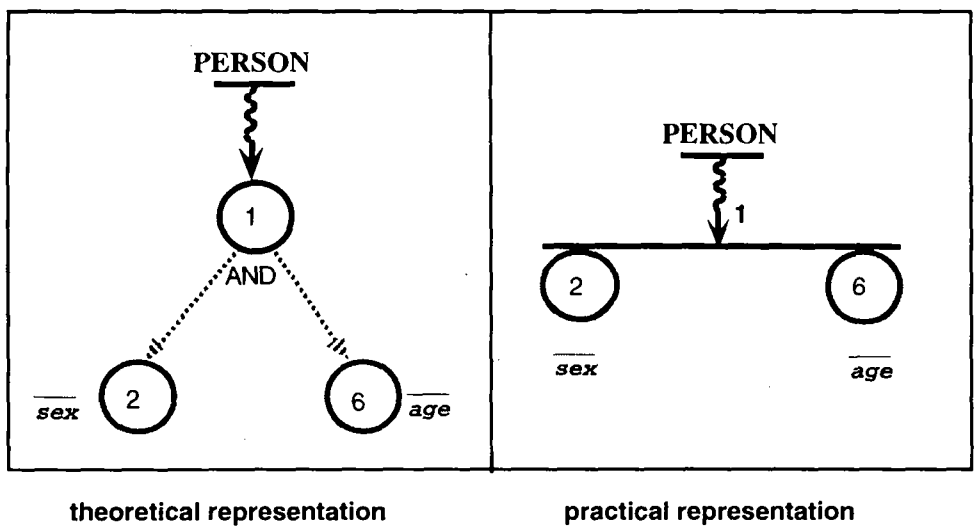


Figure 4.

Of course, all these refinements in the visual notation have a drawback : they play against uniformity, a principle which is also a fundamental one. This examplifies what is mentioned in [Green, 1989], p. 453 : *"Role-expressiveness (...) is seemingly bought at the cost of other virtues, especially uniformity, learnability and re-usability".*

3.2 color graph implementations

To complete the description and show how it is expressive, let's mention briefly how methods and memory representations attached to a color graph can be represented in a visual way.

3.2.1 memory representations

A memory representation can be attached to each color (in a c-graph) or each pigment (in a p-graph) of a dimension (provided this dimension is not explicitely or implicitely defined as abstract[9]). This memory representation is made of a few **cells** (i.e. instance variables in Smalltalk, slots in CLOS). Like transitions, memory representations are inherited along reflex transitions (except the decomposition ones). When no memory representation is explicitely defined for a chroma, its own memory representation is obtained by inheritance (and possibly combination). When a local memory representation is defined, this one can be combined or not combined with the possibly inherited one(s) : in final, this depends on the user's choice.

To visually express these roles, memory representation is shown by a pattern inside the chromas in question (if a color display is used, a different color may be used in place of a pattern or as a complement). When no memory representation is defined for a chroma, this one is white. An abstract chroma may be dimmed.

3.2.2 methods

Two methods may in principle be attached to one transition : a **pre-method** at the transition source ; a **post-method** at the transition destination[10]. These methods are collectively termed **micro-methods** and are represented by two small circles : a white one at the source ; a black one at the destination. Like transitions and memory-representations, micro-methods are inherited along reflex transitions (except the decomposition ones). See [Borron, 1996e] for more details.

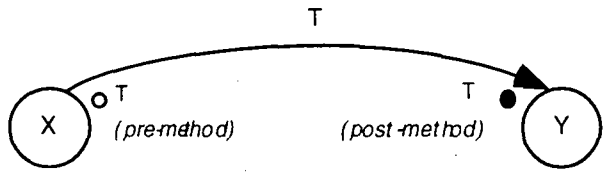


Figure 5.

Frequently enough, a pre-method alone convenes : this matches the OOP case. Sometimes, a post-method alone is a better choice (see next example) ; other times, a couple of pre- and post-methods offers is better expressive. (This third case is demonstrated with a *Window* class having a color *iconified* and a color *opened* (see [Borron, 1996e]). The *open* transition from this first color to the second one is obtained with a pre-method for erasing the icon and a post-method for displaying the window itself (the *iconify* transition does the opposite using also a couple of pre- and post-methods). Such a neat modelling cannot be rivalled by traditional OOP.)

Note that all micro-method bodiess are purely sequential pieces of code : no tests exists in their bodies ; looping is systematically obtained by recursion. Compared to more traditional languages, COP externalizes all test control structures in data structures : partly in the class hierarchy (as OOP), partly in color graphs. This can be understood as a choice in favor of more expressiveness.
The simple representation of micro-methods adequately refers to their intrinsic simplicity. The programming environment is supposed to display their code when double-clicking on their bubble representation. A separate report will describe this environment.

3.2.3 example

Next figure illustrates the visual notation used for implementing a color graph, here for a simplified *Stack* class.

For the sake of the demonstration, different memory representations are considered for a *Stack* instance depending on its current state: (a) no cells at all in the *empty* color ; (b) an *elements* list in the ephemere color ; (c) no explicit specification in the *not empty* color. Thus, the c-graph is displayed with a different pattern in colors 1, 2 and 3. In node

---

9 The *object* dimension of class *Object* is explicitely abstract ; the *LIFO* property (dimension) of class *Stack* is implicitely abstract ; the *bounded* dimension of the *Bounded* mixin is implicitely abstract...

10 In fact, several pre- and post-methods may be attached to one transition since COP supports sophisticated combination methods using qualifiers, fully generalizing the CLOS combination methods (cf. notably, the *:before*, *:after*, *:around* and primary methods of the standard method combination of CLOS).

2 (*not empty*), the white pattern means the memory representation is inherited from the ephemere node 3. In node 1 (*empty*), this same memory representation is also inherited, but —in this case— it is masked by the one defined locally.

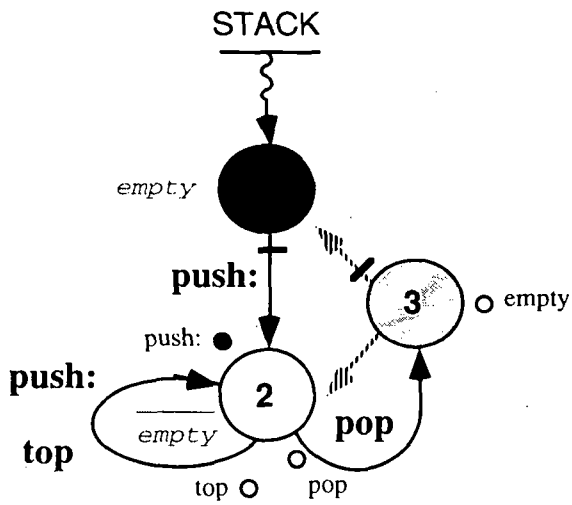


Figure 6.

Using the *Stack* memory representations just described, a pre-method appears absolutely useless to implement the *push:* transition in the *empty* color (a cell is needed to hold the pushed element, but this cell does not exist in the *empty* color) : a lonesome *push:* post-method is thus used. This post-method is also used for implementing the circular *push:* transition in color 2. Other methods are pre-methods (pre-methods correspond to usual methods in traditional OOP). The *empty* pre-method attached to the ephemere node 3 is the one which is automatically triggered by the underlying system to make a choice between nodes 1 and 2 after a *pop* message. Constant micro-methods may be directly generated from the conditions attached to these nodes : such methods are shown only on user's request (they are not shown in the figure below). The two small bars (between nodes 1 and 2, and between nodes 3 and 1) represent automatic changes of representations.

As enounced above, all micro-method bodies are purely sequential pieces of code. For example, in a more traditional language, the *pop* method tests whether the *Stack* instance contains a number of *elements* or none ; then it does the actual job of *popping* the *top* element, a purely sequential activity. Here, the test in question is not done as such since the *pop* transition is valid only when the *Stack* instance is *not empty* . When existing, the *pop* transition has the effect of running the *pop* pre-method (which does the actual job) and then moving the instance token to the ephemere color 3 ; no post-method exists (this is equivalent to a void body) so, at this moment, the underlying system automatically tests the instance vs. the conditions of the two possible selection destinations (*empty* or *not empty*) ; finally, the instance token is moved to the adequate color. This participates to maintaining the instance token in a color that always accurately reflects the actual instance state. To summarize, the test which is normally inside the *pop* method body in traditional programming is here absent from the *pop* micro-method bodies, but it is externalized in the c-graph : it is present via the selection construct, which one is perceived as more expressive than an "if-then-else" traditional control structure.

This partly responds to the remark made by T.R.G. Green in [Green, 1989] : *"(...) although the purpose of object-oriented programming is to clarify relations between parts of programs, one finds that relationships between methods are obscure. The only browsers provided [in Smalltalk-80] operate on explicit inheritance or message-passing links in the code, but the relationship lie much deeper"*. Report [Borron, 1996e], which somehow remodels class inheritance, prolongs our response in a profound way. This theoretical work is a serious attempt to fully answer the questions raised by Green's remark. Basically, states, transitions, methods and memory representations are all (mathematically) interpreted in a N-dimensions space (part 2 introduces this approach) ; each micro-method is sequential ; all micro-methods are combined in a declarative way (method bodies are never to be scanned for knowing how they effectively combined) ; combined methods respect monotonicity vs. each dimension (once a programmer understands —vs. each dimension— the global behaviours of the superclasses of a class , this programmer can infer the global behaviour of this class simply by composing, dimension by dimension, its incremental behaviour with the global behaviour of its superclasses). This latter point attenuates or avoids the disastrous effects of the "**long-range dependencies**" [Green, 1989], p. 448)). For more details, refer to the cited report. Much work needs to be done to build a programming environment that fully takes advantage of the proposal. An analysis of this foreseen programming environment in terms of cognitive dimensions [Green, 1989] [Green, 1990] will certainly be a major factor of effectiveness.

4. FIVE PRACTICAL PRINCIPLES

In this section, five practical principles are proposed for designing color graph instances. These are established for facilitating the mental activity of a programmer using such a visual formalism. At times, these principles also justify part of the formalism itself. Role expressiveness is often a reason motivating a choice here too. For two of these principles (the "avoid clauses" and the "non ubiquity" ones), one can soundly argue that they instantiate role expressiveness.

4.1 The "unique destination" principle

4.1.1 Rationale

The "unique destination" principle is the basis of the selection construct. The idea behind it is some form of simplicity and regularity : given a chroma, the programmer is asked to *"specify one and but one destination per message"*. If several destinations are possible, a fictious destination is to be created (it is an ephemere chroma) and the underlying system is taken advantage of for choosing between the real destinations by automatically testing their conditions. This contrasts with a possibly more immediate representation : several regular transitions outcoming from the same source node, labelled by the same message selector and flowing to several different nodes. In our formalism, a single regular transition is used instead which flows to the ephemere node.

4.1.2 Example

The *Person* color graph (figure 1) features four such examples : first two correspond to initializations (one for choosing between *male* and *female*, one for choosing between *child, teenager* and *adult*) ; last two, to the incrementation of *age* by *have-birthday* (choice between *child* and *teenager* on one hand, between *teenager* and *adult* on the other hand).

4.1.3 Alternative

Our "unique destination" principle conflicts with one that could be named the "economy" principle (this principle is studied in subsection 4.3). Besides one ephemere node (one label and possibly a condition) and one regular transition (one arrow and one label), we also draw *S* reflex transitions (one arrow, no label) for each selection on *S* nodes. This is to be compared with *S* regular transitions (hence, *S* arrows and labels), no ephemere node and no reflex transitions. In some cases, it may be not so clear that the "unique destination" principle is advantageous. For example, the two selections for *have-birthday* in pigments 10 and 11 may appear questionable to some programmers. They may well be tempted to draw two transitions from each pigment instead. Next figure shows the result.

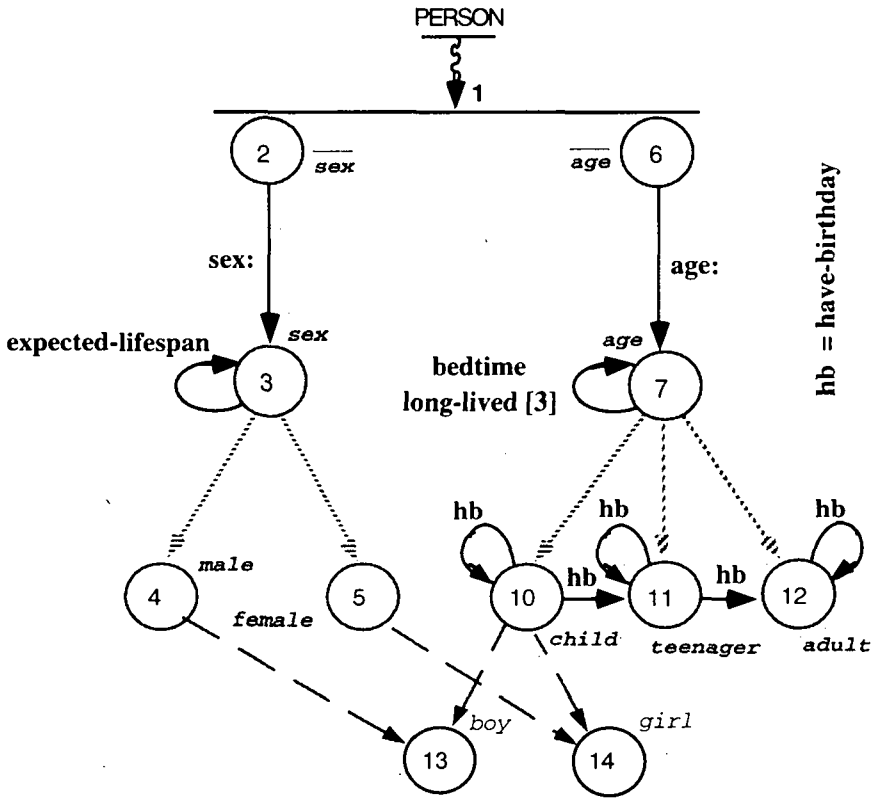


Figure 7.

4.1.4 Discussion

Personnaly, we are not convinced that figure 7 is effectively "simpler" than figure 1. This is probably because each basic element of the drawing (label, arrow, node) is not considered by itself but integrated in more high-level mental structures (once we get used to the formalism). For example, node 8 with its label and its two oucoming transitions form a single unit (a selection) ; a regular transition with its plain arrow and its label, too. Concerning the two selections for *have-birthday* in pigments 10 and 11, we thus count four elements in figure 1 and also four elements in figure 7. In addition, for a trained person, the systematic use of a selection satisfies a uniformity principle enabling a fast and safe recognition. Figure 7 does not respect this principle and hence triggers a non standard behaviour : the regular transitions need to be analysed individually before a synthesis can be made. Of course, the opposite comment can be made : someone who is used to having several regular transitions flowing out of a same node may argue that this type of representation is natural and easy to parse.

Let's suppose a demand exists for escaping the "unique destination" principle. Supporting this alternative is not a problem from an implementation point of view (either graphically or internally). From the user's point of view, it means factorization in the ephemere pigments 8 and 9 would not be possible since these two nodes would disappear (in the example, this has no consequence) ; yet, this choice is not irreversible. The most important point thus remains cognitive : it may well be better to respect a "uniformity" principle than to allow several alternatives. Our feeling is that this is not so much important for the "writer" of the color graph than for its "reader" : knowing that the "unique destination" principle may not be obeyed, the reader should make sure not to omit an outcoming transition. Hence, an additional effort of attention. The underlying system may well draw the reader's attention when and where this principle is not obeyed, but at the expense of some extra complexity ; a different and/or complementary solution is possible : an automatic transformation of the color graph so as to respect —on the reader's demand— the "unique destination" principle.... or its alternative. No decision has been made for the moment. The help of ergonoms may prove useful.

4.2 The "unique source" principle

4.2.1 Rationale

The idea behind this principle is also some form of simplicity and uniformity. It states that an acceptable message should preferably be associated with but one chroma. In other words, "*specify one and but one source per message*". In such a form, this principle is quite general. It states that **similar** transitions (i.e. labelled with the same name) flowing from different nodes should preferably be replaced by a single transition flowing from a unique node representing all the actual source nodes of the replaced transitions. This principle can thus be applied vs. a message valid for several instance states (described with different colors or p-chromas) or vs. a message valid for one state, this state being described with several p-chromas.

4.2.2 Factorizing in ephemere nodes

Next figure expresses the basic idea of this principle. A transition *T* is flowing to destination *d* from source *a* as well as from source *b*. Creating or reusing a node *c* factorizing the two source nodes enables the factorization of the two transitions *T* into a single one from *c* to *d*. (The transition in *c* is termed the **factorizing** transition ; the transitions it replaces in *a* and *b* are termed the **factorized** transitions.)

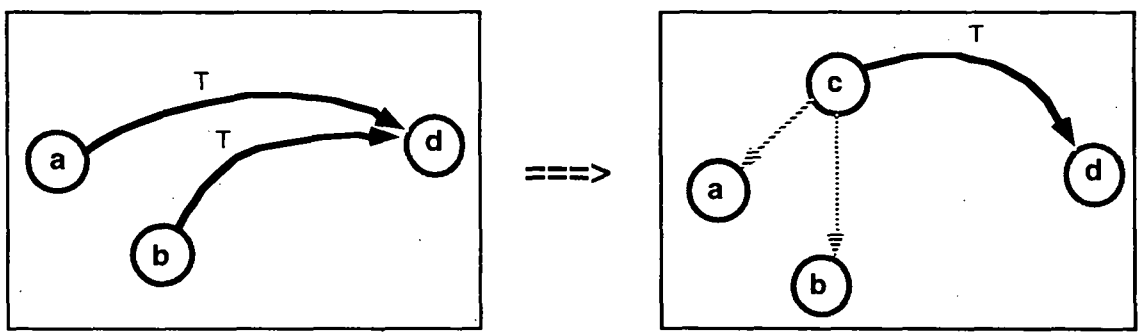


Figure 8.

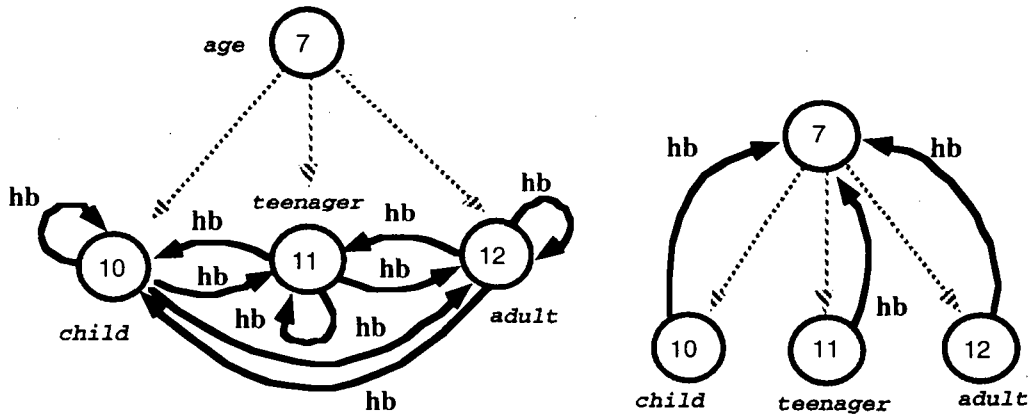
The creation of a brand new node *c* (termed a **virtual** node[11]) for factorizing a *T* transition is possible. In this case, this virtual node *c* and the reflex transition (to be created too) that flow from it to nodes *a* and *b* instantiate a **factorization construct**. Note that no token nor mini-token may flow thru a virtual node : a factorization construct is a purely static construct.

The case of reuse is cheaply brought by selections : node *c* is the ephemere node ; transitions attached to it are inherited along the reflex transitions. Thus the reflex transitions are used both dynamically (the token or mini-token moves along one of them when the selection fires) and statically (for inheriting the factorizing transition at the selection destinations). Note there is no ambiguity about the factorizing transition : this transition cannot mean a message (i.e. an external event) can be sent to the instance when in the ephemere node since, when in the ephemere node, the underlying system is only giving attention to its own responsability, i.e. deciding which reflex transition to fire (depending on the conditions of the destinations). No external message can be sent to the instance in this "state". Thus, the meaning of a transition attached to an ephemere node is clear : it is a factorizing transition. (This clarity is increased by the decision to not show the testing transitions.)

In practice, factorization constructs are ruled out : they play against the "economy" principle (addition of one virtual node plus several reflex transitions : this can clutter the color graph) ; they are useless at run time (see above) ; their structure, a diverging one like the selection, is a potential source of confusion (cf. subsection 3.1). On the opposite, factorizations in ephemere nodes are extremely frequent.

4.2.3 Example 1

To illustrate this with a typical example, let's consider once again the *have-birthday* transition in the *Person* color graph. Let's suppose we do not care about a precise modelling nor about an efficient implementation, but do prefer a modelling simpler than the ones presented in figure 1 or in figure 7 : we simply want to say that *have-birthday* changes the *age* of a *Person* instance. From a modelling point of view, this means that three *have-birthday* transitions flow out of each category (*child*, *teenager* and *adult*). Hence, nine transitions (see figure 9). First of all, the application of the "unique destination" principle to the set of three *have-birthday* transition flowing out of each category reduces the nine transitions to three, all flowing to the ephemere node *age* (see figure 10). Then, applying the "unique source" principle to these three transitions further reduces these three transitions to one (see figure 11). Note the *have-birthday* label is underlined. This is to avoid a confusion with another sort of factorization which is quite frequent :*bedtime*, *long-lived*, *expected-lifespan* are such. The factorized transitions, in these cases, are circular (destination = source). In technical terms, *bedtime* is termed **i-circular** ; *have-birthday* , **g-circular**.



Figures 9 & 10.

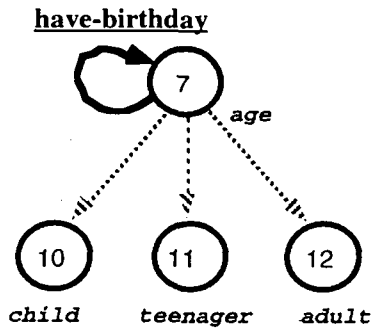


Figure 11.

[11] Formally, a node is either a chroma or a virtual node. In practice, "node"is often used in place of "chroma" since virtual node are not used.

4.2.4 Example 2

The factorization in an ephemere node is a powerful technique even in cases more complex than the one just studied : it involved a single dimension ; so, let's now consider an example involving several ones.

For this, let's model the *boy* and *girl* testing transitions. By definition, blend 13 (resp. blend 14) is such that *boy* (resp. *girl*) is true. The condition *boy* (resp. *girl*) is false in blend 14 (resp. 13) as well as in all other fully initialized states. If we want to represent the testing transition *boy* (or *girl*), we must attach it to all fully initialized states –and not only to the *boy* (or *girl*) node. The problem is that no unique chroma exists in figure 1 for representing all these states : if one existed, then attaching it the *boy* and *girl* transition would satisfy the problem (and the "unique source" principle).

Two solutions using an extra node will be shown in a subsequent section (namely, in subsection 4.3.3).

Let's propose another solution. Unfortunately, it a priori violates the "unique source" principle. It consists in splitting each transition (*boy* and *girl*) in two parts, termed **micro-transitions** : one attached to node 3 (for factorizing the micro-transitions attached to nodes 4 and 5) ; one, to node 7 (for factorizing the micro-transitions attached to nodes 10, 11 and 12). These two micro-transitions should be reminiscent from the fact that they form a unique transition : this coupling is obtained by attaching them a clause referring to the other pigment. (If unconstrained, a micro-transition in pigment 3 (resp. in pigment 7) would be valid also for any pigment of the other scale, notably in pigment 6 (resp. in pigment 2) which is not correct : both initializations should have occurred.) The pair of constrained micro-transitions constitutes a **multi-micro-transition**. Next figure shows the result.

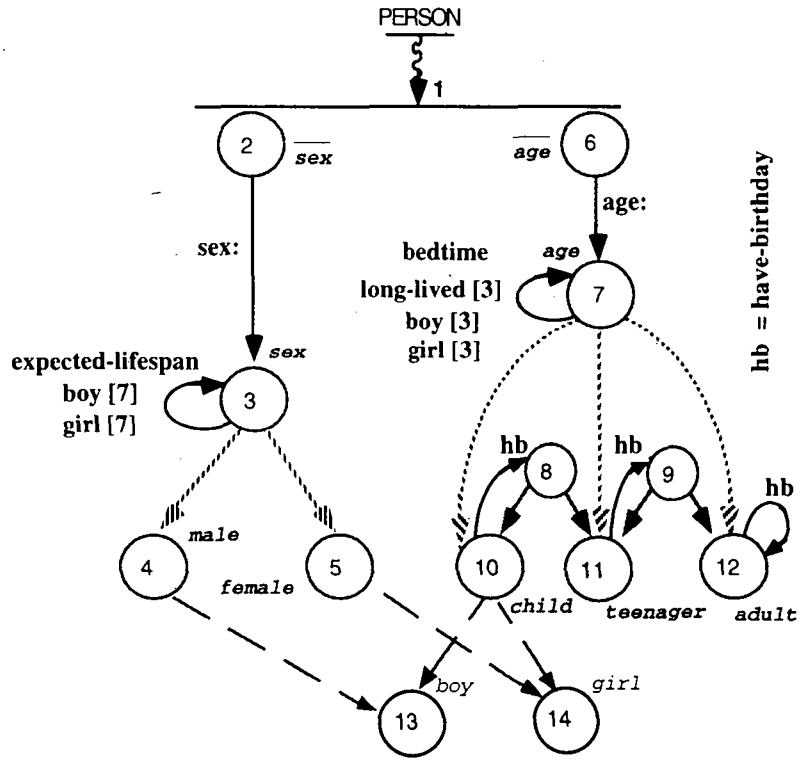


Figure 12.

As a matter of fact, this last representation can be simplified. Given that the *boy* (resp. the *girl*) testing message is side-effect free, the two *boy* (resp. *girl*) micro-transitions are circular (more precisely, i-circular). Thus, the "unique source" principle can be obeyed by keeping but one constrained circular micro-transition (in either 3 or 7). Hence, in final, a representation for *boy* and *girl* transitions similar to *long-lived*, a (simple) constrained transition[12].

[12] The same reasoning could have been made for *long-lived*, but the solution being already known, a different example was considered as preferable.

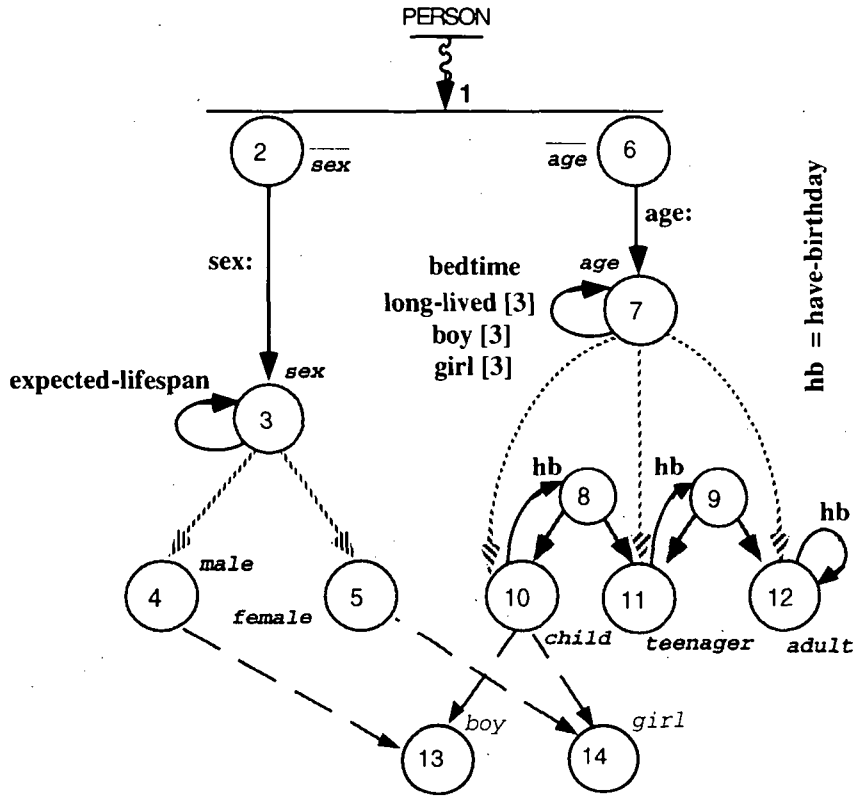


Figure 13.

4.2.4 Conclusion

With this second example, we discover that the "unique source" principle was in fact systematically applied, yet implicitely, for producing the apparently simple transitions *age:*, *sex:*, *expected-lifespan*, *bedtime*, *have-birthday*, *long-lived*, etc. Each one (for example, *sex:)*, being valid for a priori any pigment of the other scale and having no side-effect on it, has in fact a circular counterpart (i.e. a micro-transition) in each one. (By factorization, almost all these circular counterparts disappear, leaving a micro-transition in a few pigments only : in 6, to take the same example.) More generally, each message is theoretically described by its effect on each dimension, with possibly several micro-transitions for each one (as just shown in the *sex:* example). By convention, to satisfy the "unique source" principle, circular micro-transitions are normally eliminated from multi-micro-transitions ; nevertheless, one micro-transition should at least subsist : possibly constrained, this one is termed "the" transition.

Can we get rid of multi-micro-transitions in any case ?

— Conditional transitions are to be distinguished from other transitions. First, conditional transitions are normally not shown : the programmer is thus not bothered by them. Second, they can always be transformed into i-circular, possibly constrained, transitions. Third, their case is somewhat special since the existence of a condition in one blend propagates[13] to a number of other states which are not always materialized by blends (this is typically what occurs in the *boy* and *girl* examples) ; yet, the problem disappears because the evaluation of conditions is side-effect free (hence, the result stated in the second point).

— Now, let's turn to regular transitions which are not generated from conditions. These are managed directly by the user. Theoretically, the color graph formalism enables all of them to be represented simply, using blends and selection on blends if necessary. Yet, because the number of possible blends dramatically increases when the number of dimensions increases (combinatorial explosion of states), the programmer may well –in practice– be constrained to refrain from using (too many) blends ; hence, some transitions cannot be implemented without a priori resorting to multi-micro-transitions. Even if it is possible to abstractly construct especially "bad" examples, the problem is quite likely not extremely severe : first, circular multi-micro-transitions can always be reduced to constrained circular transitions ; second, orthogonality vs. a number of dimensions (often vs. N-1 dimensions) is quite frequent in real world. An example with a priori several multi-micro-transitions is treated in subsection 4.5.2.

---

[13] Let B the considered blend. In each pigment scale forming B, the algorithm first looks for the highest nodes (pigments) from which B derives by a reflex transition path (the roots of each p-ancestor-graph, in technical terms). Then, if several ephemere pigments were found for a same scale, the algorithm compares their offsprings and retains but the most prolific one. In the *boy* and *girl* examples, ephemere pigment 3 is retained in the first scale (unique candidate) ; ephemere pigments 7 and 8 are obtained in the second scale ; yet, since the destinations of the selection in 8 (i.e pigments 10 and 11) are all part of the destinations of the selection in 7 (i.e pigments 10, 11 and 12), the ephemere pigment 8 is automatically eliminated. Thus two pigments are retained, 3 for the first scale and 7 for the second : states that are concerned by *boy* and *girl* all derived from them.

4.3 The "economy" principle

4.3.1 Rationale

We already mentionned this principle in subsections 4.1.3 and 4.2.2. This is a very general one. It states that we better choose the cheapest possible representation. As examplified in the cited subsections, the cost to be taken into account for a given color graph is evaluated in terms of displayed graphical elements (labels, nodes, arrows,...). Besides its simplicity, the justification for such a coarse evaluator is that, in our view, any element requires some effort to be analysed and placed in the data structure our mind elaborates, whatever this data structure is. This data structure is likely to be expensive to build and complex in its details if the color graph is itself complex. At least because the evaluator is coarse and because the complexity it measures is not likely to relate linearly with the operations our mind carries out when reading and exploiting a color graph, the "economy" principle should not be used to cut short between two alternatives, especially when it conflicts with other principles. It is nevertheless quite useful for a programmer attempting to define the simplest possible color graph for a class. The "economy" principle is the one which justifies partly our decision for not adopting the factorization construct (cf. subsections 3.1 and 4.2.2), and fully our decision for not representing conditional transitions : these are usually numerous and can clutter up a color graph with circular transitions that are in fact easy to infer from the conditions. The "economy" principle is also the one that underlies factorization and hence local inheritance rules. To some extent, it also underlies the class inheritance rules.

4.3.2 Example 1

Let's see how a programmer may refer to the "economy" principle. We abandon for a moment the *Person* example to focus on a different class, the *Circle*. A first color graph is defined for it, a c-graph. Four colors are used to represent the four possible states of a *Circle* instance : an uninitialized one (color 1) ; two partially initialized ones (colors 2 and 3) ; and a fully initialized one (color 4). When uninitialized (color 1), two messages are acceptable for initializing either the *radius* or the *center* of the instance, hence two transitions (*radius:* and *center:*), each one leading to a new color. When its *radius* is initialized (color 2), the instance may be asked to return its value or to change it ; the instance may also be asked its *surface*. These three messages do not modify the current color. On the opposite, a *center:* message leads to color 4 and a *radius-* message, to color 1 (this message uninitializes the instance *radius*)... Once colors 3 and 4 have been examined in the same way, we get the following figure.

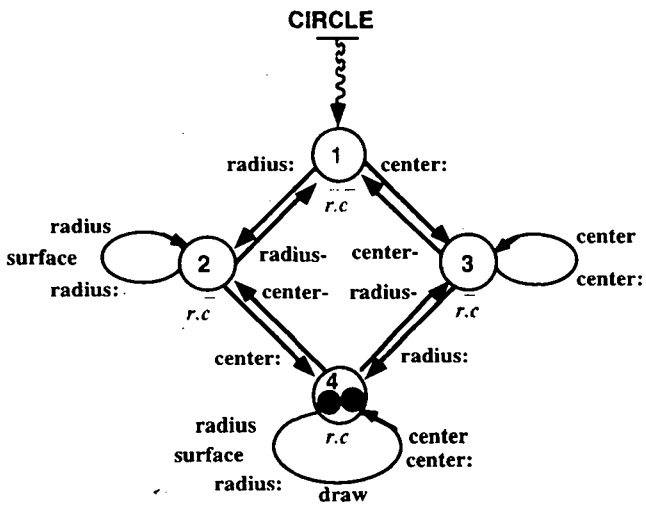


Figure 14.

Clearly, such a color graph is likely not to obey the "economy" principle : are attached to color 4 the groups of transitions attached to colors 2 and 3 : only one transition (*draw*) is specific to color 4 ! In addition, transitions along paths 1-2 and 3-4 are identical ; same remark for transitions along paths 1-3 and 2-4. Can't we get rid of all these repetitions ? Next figure shows a solution.

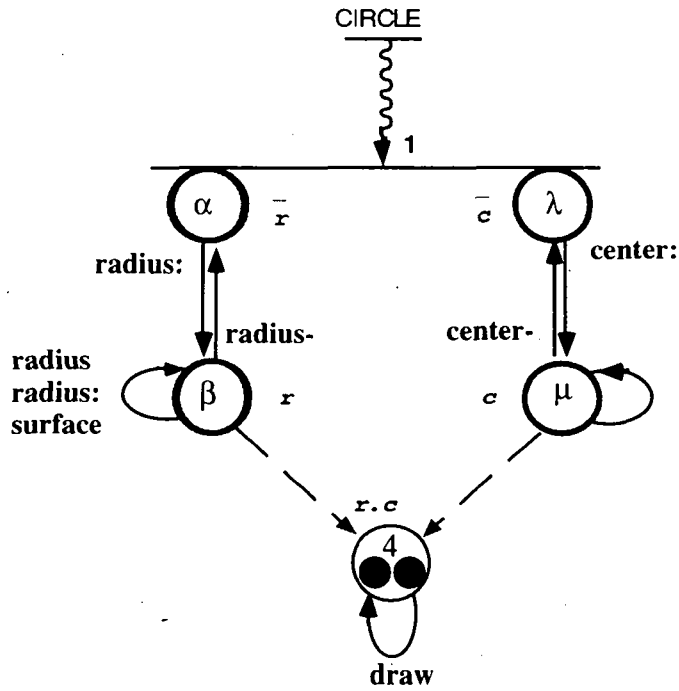


Figure 15.

The *Circle* interface is now depicted as a p-graph : two dimensions are used to describe the behaviour of a *Circle* instance, a *radius* and a *center* one. The improvement is noticeably important : the number of displayed elements has decreased ; being simpler, the graph is easier to read and grasp. This type of improvement gets in fact more and more important when the number of dimensions gets higher and higher (a c-graph is potentially a victim of a combinatorial explosion of nodes).

4.3.3 Example 2

Let's come back to the represention of the *boy* and *girl* testing transitions (subsection 4.2.4).

a) Solution 1 : super-blend

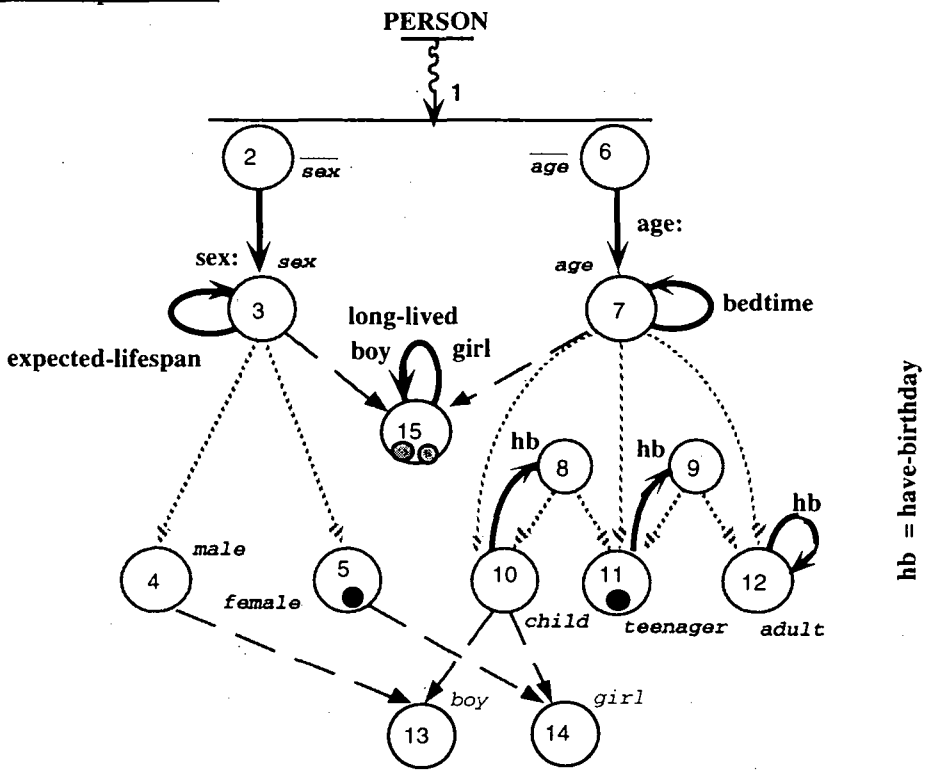


Figure 16.

To respect the "unique source" principle, a possible idea is to create a node as a conjunction of nodes 3 and 7. See next figure. This new node is termed a **super-blend** since it represents a set of blends. When the current instance state corresponds to one of them (ex. : when the two mini-tokens are in 13 or in (5 11),...), the transitions attached to the super-blend are also valid (this is materialized by two special mini-tokens drawn in a blurred way inside it). Note that *long-lived*, which is valid for any fully initialized instance, could well be attached to the super-blend. This solution is expressive ; yet, the added matter (one node and two reflex transitions) plays against the "economy" principle. Compared to the constrained transition solution (figure 13), this proposal is less economical, but much more expressive.

b) Solution 2 : constrained subtree

This solution is brought from our study of mixins ([Borron, 1996d], §.3.2.2.2). A constrained subtree is a fairly general useful abstraction. The subtree is displayed under the decomposition bar and, above it, a clause (between brackets) is shown. This clause specifies when the subtree is valid : its contents is a chroma –or, more generally, a list of chromas. When the instance state meets this chroma –or, one of them, the mini-tokens that mark the instance in this chroma appear in the root of the constrained subtree.

Like the preceding solution, this one respects the "unique source" principle. It is better from the "economy" principle (no reflex transitions) ; yet, it is more abstract due to the clause. Compared with the constrained transition solution (figure 13), this one is less economical and more abstract (its clause is more complex).

However, the merit of this solution is to separate from the rest what requires a non trivial condition on both dimensions. The modelling gets easier to parse : all transitions below nodes 2 and 6 are inherited in a simple way (top down parsing). The difficulty, if any, is isolated in the righmost part and the user, knowing that, can adopt an appropriate working strategy. In an interactive environment, the constrained subtree can be displayed only when valid, thus liberating the user from checking its clause. Finally, this modularization indicates well how a class can be splitted up for taking advantage of class inheritance : except for the blends, each subgraph corresponds to the potential color graph of a superclass ; the blends and the constrained subtree(s) correspond to the increment that will remain attached to the composed class. In the considered example, one can extract this way a *Sex* class as well as an *Age* class, define the *Person* class as inheriting from them and as featuring also what requires both dimensions (the *boy* and *girl* blends, plus the *long-lived* regular transition -without forgetting, of course, the *boy* and *girl* testing transitions) : this is exactly what is done in [Borron, 1996c].

In this line, a constrained subtree may be used more generally to zoom on subparts of a color graph when appropriate, for example when debugging a program. Its reciprocal is the possibility to abbreviate a subtree as a single node (holophrasting), a dynamic instantiation of the "economy" principle so to speak. Examples are given in the cited reference.

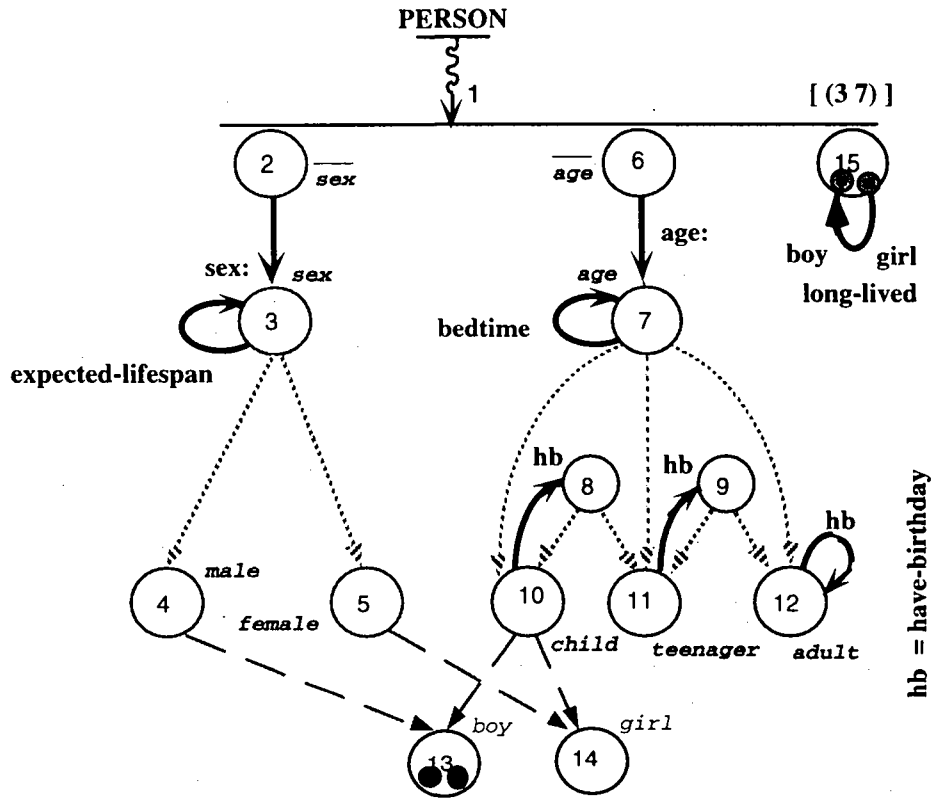


Figure 17.

4.4 The "non ubiquity" principle

This principle states that the current instance state should not be represented more than once in a color graph.

4.4.1 First example

One may argue that the first *Circle* p-graph (figure 15) is somewhat ambiguous in the sense that the state "*radius* and *center* initialized" is represented twice, once by the pair of pigments (β μ), once by blend 4. As a matter of fact, no ambiguity exists since an instance state is precisely defined in a p-graph by the position of the mini-tokens. By definition of the conjunction construct, both reflex transitions β-4 and μ-4 fire once a second mini-token arrives in β (or μ) while the first one is already in μ (or β) : both mini-tokens move to blend 4 ; being not ubiquitous, they are not at the same time in blend 4 and in pigments (β μ). This is clearly indicated in the figure : the two mini-tokens exist in blend 4, not elsewhere. The "non ubiquity" principle is obeyed. Yet, the *Circle* instance in blend 4 inherits each group of transitions attached to pigments β and μ as a contribution of each instance dimension (here, an instance part) considered in an isolated manner.

4.4.2 Second example

Let's come back to the super-blend solution used for representing the *boy* and *girl* testing transitions (subsection 4.3.3). This representation seems to suffer from a major drawback : a fully initialized state is apparently represented twice in the color graph. (The two mini-tokens marking a state are apparently duplicated : for example, they exist in this new node and node 13.) As a matter of fact, the "non ubiquity" principle is not invalidated : the status of the mini-tokens in the super-blend is not exactly the same than the status of the original mini-tokens. The formers do not indicate the current state, but the belonging of the current state to a set of states having specific properties ; in addition, they do not exist systematically as the original mini-tokens. Such mini-tokens in the super-blend are termed "**traces**" or "ghosts" : vs. the original mini-tokens, the traces are like slaves vs. masters. This is why they are drawn in a blurred way.

4.4.3 Third example

Here, we consider a different example illustrating the possibility given to the user to pass from one system of dimensions[14] to another one. Next figure shows one case : in the first system, a *Point* instance is created by *make-cartesian* : once fully initialized, the instance can be considered in cartesian or polar coordinates ; in the second system, a *Point* instance is created by *make-polar* : once fully initialized, the instance can also be considered in cartesian or polar coordinates. As a matter of fact, the memory representation effectively used in the first system and/or in the second one may be polar or cartesian. Yet, in case an optimzed representation is known to be attached to each system, the user can take adavantage of this knowledge for the computations to be done by explicitely moving the instance from one system to the other one. The underlying system automatically takes care of the representation change (if any)[15]. It is clear, in this case, that the pair of mini-tokens is moved from one system to the other and never exists at the same time in both systems : the "non ubiquity" principle is thus satisfied.

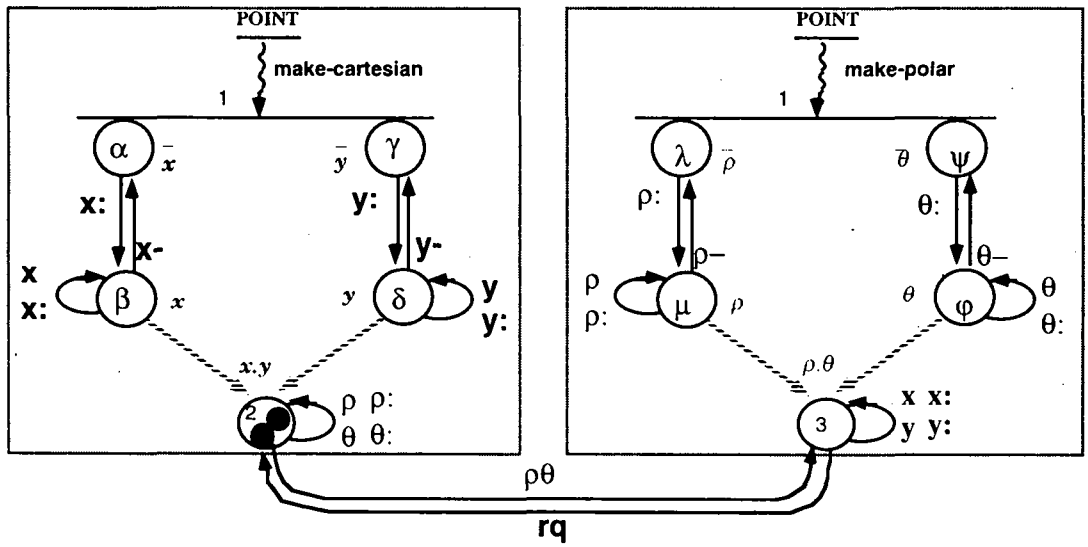


Figure 18.

---

14 To keep the metaphor of colors, a system is termed a palette.

15 This is analogous to a **change-class** in CLOS, yet done automatically.

Note that the possibility of passing from one system to another one can also be taken advantage when bootstrapping a system, for example one implementing COP.

4.5 The "avoid clauses" principle

While we are not completely sure that psychologists effectively acknowledge the quality of persons for being preferably "auditive", "visual" or "kynesthesic", we personnaly feel much more comfortable with a visual representation (which we can animate in our own mind, in fact) than with a formal, textual representation. This may be why we would strongly tend to promote as a principle : *"avoid as much as possible constrained transitions and especially multi-micro-transitions".*

In practice, this means *"preferably attach a transition to the chroma that thoroughly represents its source state"*. When a constrained transition exists, the reader may be given help first in distinguishing it as special, second in identifying the pigment/blend specified in the clause : the underlying system may, for example, highlight this node if the user clicks on the clause. It may even propose an equivalent p-graph (on user's request, possibly using a list of preferences stated by the user before hand).

4.5.1 Example 1

Let's come back to the *Circle* example. When a *Circle* instance is fully initialized, i.e. when both its *radius* and its *center* are initialized, a *draw* message may be sent to it. How can we depict the corresponding transition ? To which node should we attach it ? The solution we chose in figure 15 consists in materializing the fully initialized state using a conjunction. The *draw* transition requires both β and μ pigments in a symmetric way : the chosen representation does not break this symmetry.

Another representation is possible : it consists in attaching the *draw* transition to either the β or μ pigment, and to make it constrained by the other pigment. This representation satisfies the "unique source" principle but breaks the symmetry[16]. Next figure shows one of the alternatives.

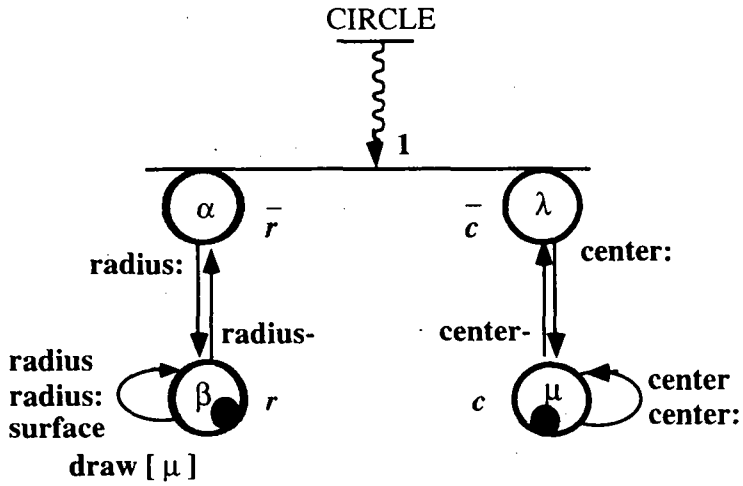


Figure 19.

Both solutions respect the "unique source" principle ; and, from the "economy" principle point of view, the last one is a priori better that the one shown in figure 15 (addition of one node and two transitions). However, our feeling is that the last representation is less immediate to understand than its alternative. The first representation of figure 15 is clearly more expressive. The last one requires from the reader an extra effort to take into account the clause *[μ]*. (It seems likely that the user tends to browse a bit the *center* scale so as to locate the pigment in question. In such a case, the underlying system may automatically propose the first p-graph (on user's request).)

Note also that the existence of blends also enables to materialize conditions like *boy* or *girl* (see figure 1, for example) whereas such conditions would otherwise remain only abstract synonyms for *child.male* or *child.female*.

---

[16] To keep the symmetry, one may choose to use a multi-micro-transition, i.e. to have one circular constrained micro-transition, *draw [μ]*, flowing of out of β and another one, *draw [β]*, flowing out of μ. This solution is not really satisfying : it does not respect the "unique source" principle and requires in addition, from the reader, an analyiss to detect that the two micro-transitions have the same meaning.

However, one can say that blends are potentially dangerous. This refers to the risk of combinatorial explosion when the number of pigments in each scale (and/or the number of scales) increases. This remark requires a response. As such, it is true. However, the important word here is the adverb "potentially". In this example as well as in the *Person* example, the number of blends that are effectively instantiated is quite small. As shown by the *Person* example, all potential blends are not to be instantiated. (The number of blends in this example is not higher than in the original *Person* example of [Chambers, 1993] : in the cited paper, on page 286, the *boy* and *girl* classes were defined too[17].) The blend feature is only an alternative enabling simpler specifications in usual cases as shown here (it avoids clauses), a mere possibility that is supposed to be used cleverly (when leading to too many blends, users are naturally supposed to refrain from their use). In real life, there are many things that are potentially dangerous and are used anyway (for example, subordinate sentences may be used without any limit a priori ; but, as anybody may see for oneself, this is not the case in practise.) The suppression of blends from our formalism would mean programmers and designers are not clever enough to use them appropriately. Clearly, we do not think so.

4.5.2 Example 2

Fom a conceptual point of view, the existence of a super-blend introduces some extra complexity in the graph (one node, plus a number of reflex transitions) ; at the same time, it enables a less abstract representation of a transition (one source, no clause). In the *Person* example, the difference was not crucial : a constrained transition may well be prefered to the super-blend solution.

A more difficult example is represented by the next figure : once in state (α β), a *T* transition has the effect of changing the state to either (α β'), (α' β) or (α' β'). In this case, the use of a super-blend (node δ) and the use of a **negative** transition (from γ to γ) allows an elegant and expressive solution. (The effect of the negative *T* transition is here to withdraw its own destination from the destinations of the other *T* transition which fires at the same time.) Note that this solution does respect the "unique source" principle, but not the "unique destination" principle.

The super-blend solution can be compared with its alternatives : except for the constrained subtree one (which simply moves the blend, the super-blend and the transitions out of the main p-graph), all are fairly difficult to understand. The figure afterwards shows one while, like Harel's higraphs, does not use blends : to model the *T* transition, three multi-micro-transitions are required. A convention is necessary to show the association of two micro-transitions belonging to a same multi-micro-transition : the drawing uses both the text (plain, underlined, in italics) and the pattern (resp. plain, dotted, dashed). Both the recognition and the interpretation are (much) more difficult than when using the super-blend. In such a case, the "avoid clauses" principle is better applied.

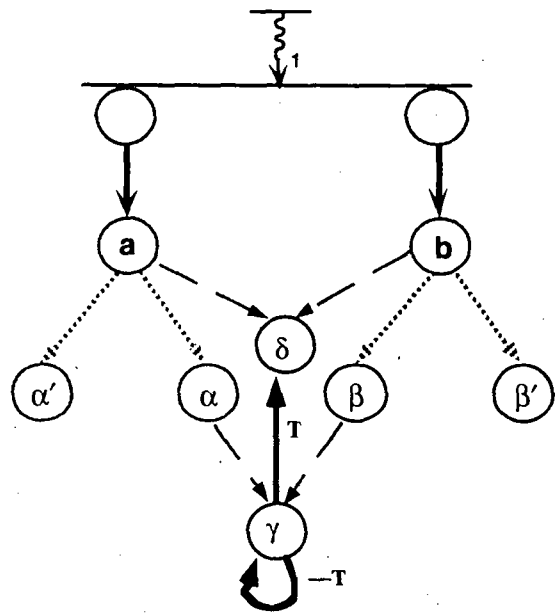


Figure 20.

17 ....without apparently provoking any alarm. The form in which a same modelling is presented, text or figure, apparently triggers different (scientific) reactions...!

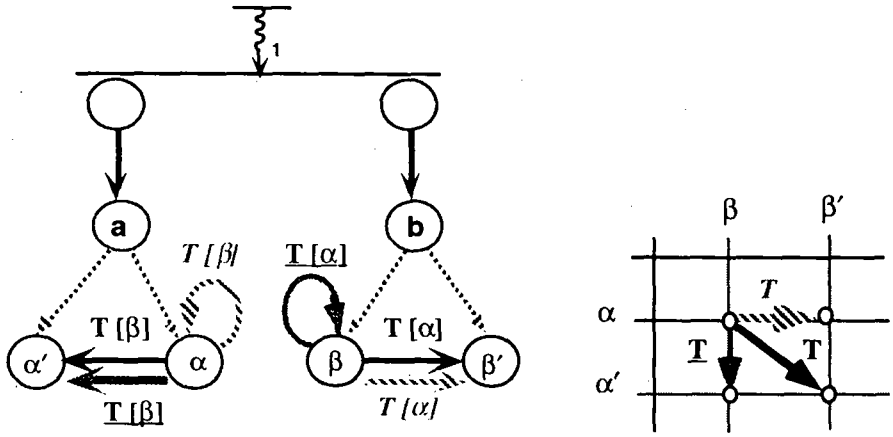


Figure 21.

Note that super-blends and negative transitions are presently a mere suggestion. If accumulated, negations may well cause understandability problems. Super-blends are also to be used appropriately when the cost/benefit ratio is manifest.

5. CONCLUSION N° 1

The design of out visual notation has been discussed in the two preceding sections, notably vs. the quite general "role expressiveness" principle. Five practical design principles have also been proposed. With the help of examples, we showed how they mould both the color graph formalism and its specific instantiations. These principles are based on our own experience, but we strongly expect them to be backed up from a cognitive point of view, either theoretically (for two of them, the "role expressiveness" principle plays this role) or experimentally.

In practice, several principles may well conflict and there are usually several equivalent ways to define a color graph describing a same specification. The habits and tastes of a color graph writer play also a role. A couple of times, it appeared that a color graph writer may well transgress some principle, which may cause a difficulty to the reader : in such cases, it was indicated that the system may provide some help to the reader.

We expect rigorous observations of ergonoms will confirm our feeling and be the inception of enhancements.

PART II : ALTERNATIVE REPRESENTATIONS

In this second part, we provide two different points of view on color graphs. The first one relates color graphs to a representation in a N-dimensions space (this is important from a theoretical point of view) ; the second one, relates them to higraphs, a (non object-oriented) visual formalism proposed by David Harel in quite a different domain[18].

6. CARTESIAN TYPE REPRESENTATIONS

The mapping to a cartesian type representation is easy to master for 1- and 2-dimensions color graphs ; it is less convenient for 3-dimensions ; and for more dimensions, it is untractable. Yet, it is useful to mention as a mental model : because we are generally used to cartesian representations for mathematical and physical applications, its use illuminates the syntax and semantics of color graphs once the mapping is clear[19].

A color graph simply expresses the possible states of a class of objects in a N-dimensions space. In such a space, each dimension of a color graph corresponds to a specific axis ; pigments of a scale are points (coordinates) on the corresponding axis ; instance states are points in the N-dimensions space ; each mini-token moves on a specific axis ; any transition maps a point into another point (or a cloud of points).

6. 1 Example1

Let's consider again the *Circle* example. It has 2 dimensions and thus maps to a cartesian plane. See next figure. The α and λ coordinates (resp. *radius* and *center* dimensions) are distinguished to express they are initial pigments. We have not directly represented the *Circle* transitions (except *draw*) but only their projections. The figure thus corresponds precisely to the p-graph of figure 15. If we had shown all transitions between points 1 to 4 (and not their projections), then the figure would have corresponded to the c-graph of figure 14.

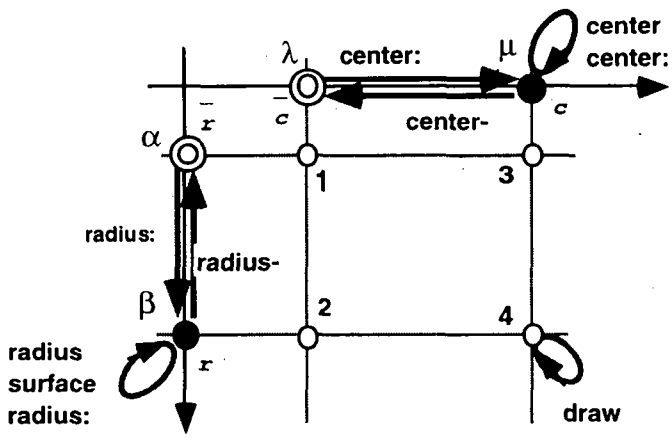


Figure 22.

This representation seconds our argumentation about the "avoid clauses" principle (cf. subsection 4.5). In the cartesian plane, the most natural way –in our view– to represent the *draw* transition is to display it as a circular edge attached to node 4. The parallel with the color graph representation is clear : we do not constrain the programmer to use only one system of presentation (the coordinates on the *N* axes or the pigments), but what expresses in the simplest way the feature (transition) to be depicted : the transition *draw* is expressed simply when attached to node 4 ; the other transitions, because of their orthogonality vs. one dimension, are so when attached to pigments (when projected on the axes). To consider an analogy, if a user manipulates a triangle, he/she may wish to consider each summit analytically (as a pair of coordinates) : this may be useful for certain operations ; but, he/she may also prefer to consider each summit geometrically (in the figure itself) for other operations.

The remark just made even extends to the point where the user may adopt a new coordinate system. For example, the cartesian coordinates *(x, y)* are rather clumsy for expressing rotations : polar coordinates (ρ, θ) are much better for this purpose. More generally, a same point in a N-dimensions space may be located in different coordinate systems. This exactly corresponds to having several pigment palettes in a color graph, a capability which is effectively supported as already shown in subsection 4.4.3.

---

18 not ours!

19 Reference [Borron, 1996e] notably uses the cartesian plane for presenting class inheritance. (Other examples are present in it.)

6. 2 Example 2

The *Circle* example is rather simple. In particular, it does not feature selections. The *Person* color graph does. So let's now consider equivalent representations of this one.

6. 2.1 Using an imaginary dimension

Let's first draw an equivalent of the *Person* p-graph (N=2) in the cartesian space : for role expressiveness, we add an imaginary dimension for selections. Next figure examplifies this addition, showing the *Person* p-graph of figure 1 in 3 dimensions (N+1).

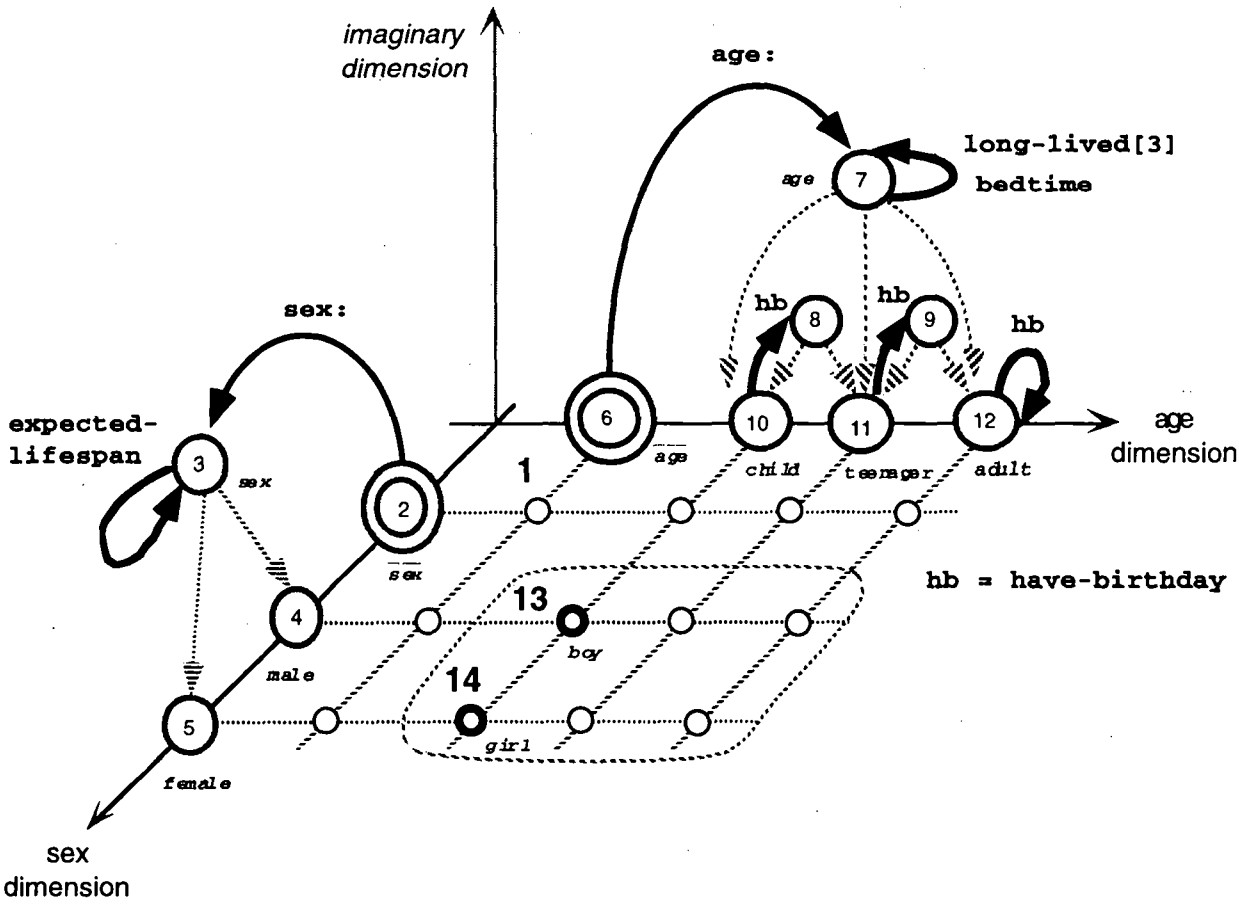


Figure 23.

The figure clearly expresses the difference between a basic and an ephemere p-chroma. A basic p-chroma belongs to the space S strictly defined by the N-dimensions ; an ephemere p-chroma, to the complement of S in the space defined by the N-dimensions plus the imaginary one :

— a basic p-chroma is either a basic pigment (like *male* and *female* ; *child*, *teenager* and *adult*) or a basic blend (like *boy* and *girl*). The first category describes an atomic substate along one single dimension ; the second one, a composite substate along several dimensions, possibly N —in which case the blend is also termed a color.

— an ephemere p-chroma is either an ephemere pigment (like 3, 7, 8 and 9) or an ephemere blend (none is shown in the figure, but *long-lived* could well be implemented using one[20]).

Because it describes a substate, a p-chroma is usually not associated to a single instance state but to a set of instance states (all those that match the associated substate). Conversely, a reachable instance state is usually not represented by a single node (as it is the case in a c-graph) but by k p-chromas, the degrees of which sum to N.

In this representation, local inheritance of transitions along the reflex transitions of selections is easy to interpret. In addition, being not materialized by reflex transitions but by initial pigments (double circles), the decomposition does not pose a problem vs. inheritance. Yet, the representation is not uniform vs. conjunctions : a conjunction is not represented with reflex transitions, but by a set of coordinates determining a point or a cloud of points.

---

[20] The *long-lived* transition, which is constrained both by node 3 (its clause) and node 7 (its source), could well be attached to an ephemere blend, the unique role of which would be to factorize *long-lived* in on place. However, this would require the six blends in question (colors) to be displayed -instead of two presently, which is uneconomical. (A cheaper solution consists in using a super-blend as done in the p-graph of figure 16.)

6. 2.2 Other attempts

The previous figure is especially clear, but at the expense of an imaginary dimension. Is it possible to get an expressive figure without that trick ? Next figure shows one possible equivalent, in the cartesian plane, of the same *Person* p-graph. Here, ephemere pigments are represented as clouds of basic pigments. As in the previous figure, we only represent basic pigments on each axis. The *long-lived* constrained i-circular transition is attached to a cloud of six points. The figure is probably easier to imagine or to draw than to read.

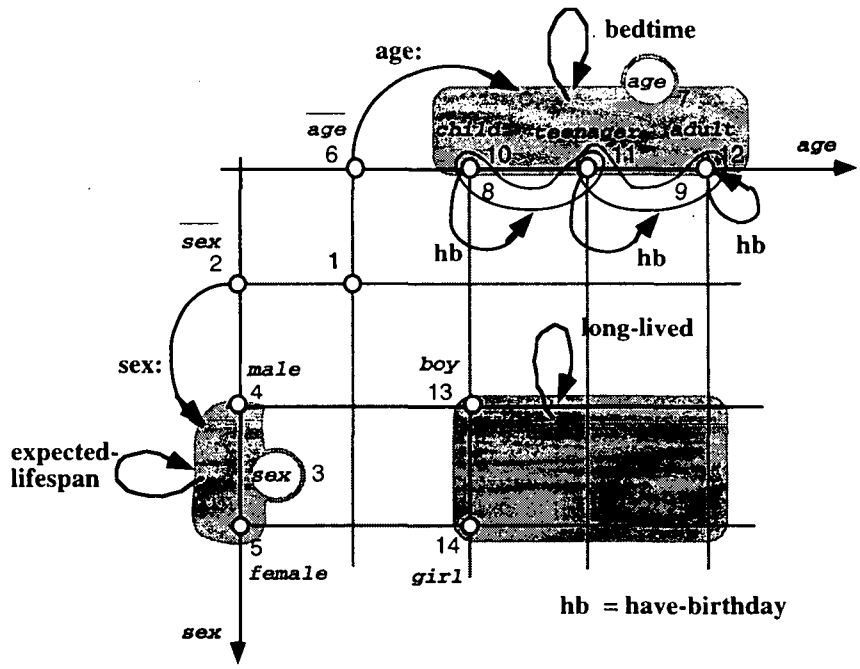


Figure 24.

An alternative representation in the same cartesian plane is obtained by replacing each cloud on an axis (ephemere pigments 3, and 7 to 9) by a point on the same axis. Compared to the preceding figure, the result is apparently clearer but less compact (see next figure). Its interpretation requires an additional search step for identifying basic points when an axis transition destination is like *"10+11+12"*. In other representations, this abstract step is replaced by a geometric connection : by an enclosing boundary in the preceeding figure ; by reflex transitions in a color graph (figure 1).

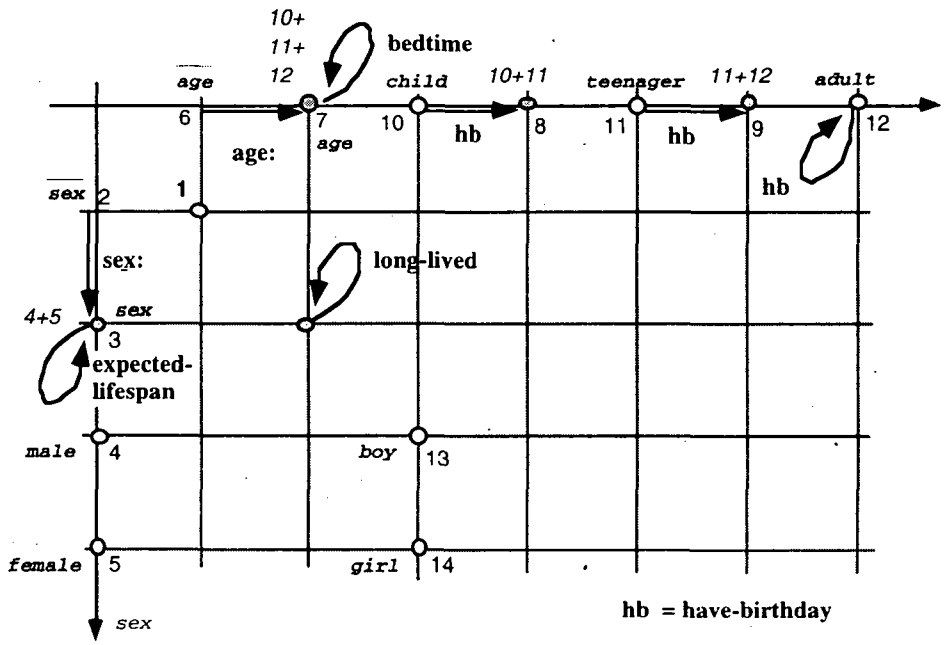


Figure 25.

In this second type of representation, a circular constrained transition (like *long-lived*) is attached to a true point (i.e. a point that is not on an axis) ; a non circular constrained transition, to a transition between two distinct true points on a same horizontal or vertical ; a (non circular) multi-micro-transition, to a transition between two distinct true points in

diagonal. This can be easily generalized to a *N*-dimensions space. (The right part of figure 21 is an example featuring three multi-micro-transitions.)

6. 2.3 Conclusion

Two points appear :
— a p-graph is really a description, in a *N*-dimensions space, of states and transitions between these states ;
— a p-graph acknowleges that a cartesian representation is usually very awkward, if not impossible, to draw.
A p-graph thus appears as a compromise between very practical constraints and a theoretical requirement (for the interpretation of class inheritance in a *N*-dimensions space, refer to [Borron, 1996e]).

7. COLOR GRAPHS vs. HIGRAPHS, or CONNECTEDNESS vs. INSIDENESS

A strong connection can be made between color graphs and higraphs (hierarchical graphs), a visual formalism proposed by David Harel.

7.1 Mapping

The key point concerns what we named reflex transitions : in color graphs, they are materialized using connecteness. Replacing connectedness by insideness remarkably leads –modulo the a priori arrangement of certain details[21]– to "higraphs" as exposed in [Harel, 1988] and its companion papers about "statecharts" [Harel et alii, 1987] [Harel, 1987].

In higraphs, colors are termed "blobs"; and pigments, too. An ephemere pigment is mapped to an encircling blob (for example, the *sex* blob encircles the *male* and *female* ones in the *Person* example). Regular transitions correspond to directed "edges" or "hyperedges". The parallel is striking when noticing that a decomposition into pigments (more exactly, into pigment scales) in COP exactly correspond to "partionning" in higraphs : in both formalisms, such features are meant to avoid a combinatorial explosion of states, an important consideration indeed.

This consideration is pushed to the utmost in higraphs : by construction, they do not allow the representation of blends. In color graphs, one has the opportunity to use them : the user is trusted not to use this facility when blends are too costly ; in other circumstances, lacking this feature means the expression of simple things gets awkward if not complex.

One should keep in mind that COP is –at least, for the moment– restricted to transformational systems whereas statecharts are meant for reactive systems, like operating or avionics systems, communication networks, ...

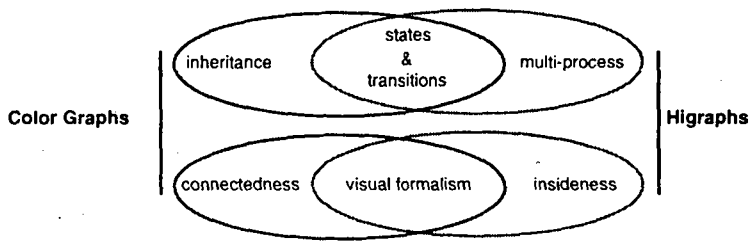


Figure 26.

7.2 Insideness vs. connectedness

As reported by references [Fitter-Green, 1979] and [Green, 1982], the insideness vs. connectedness choice has been the subject of a debate for expressing hierarchical relationships in the early seventies : [Nassi-Schneiderman, 1973] and [Jackson, 1975] had both their adherents at that time. This choice is also very clearly expressed in [Harel, 1988] where, for example, set-theoretic relationships in entity-relationship diagrams are more elegantly and cheaply expressed by insideness than connectedness. This same article (p. 522) as well as ([Harel and alii, 1987], p.55) convincingly expose how insideness can be taken advantage of for avoiding the multiplication of edges (and for expressing a notion of depth as well). They also claim this simple idea does appear as a fundamental improvement in practice *"when applied to large collection of states with many levels"* (thus confirming the "*explosive effects*" mentionned in the introduction).

Because the corresponding formalism has been succesfully applied to the behavioral specification of complex systems -notably in avionics, we must trust such a claim. Given the above comparison, the remarks are valid also for color graphs. The only differences lie in the representation of reflex transitions and in the possible existence of blends in

[21] For example, conditions are not attached to edges, but to blobs : this is natural given the underlying semantics of color graphs (all incoming edges thus share the condition of their destination blob).

color graphs. Well, blends are supposed to be used only when really few. So, the major difference —if any— would be in the representation of reflex transitions for selections : the hypothesis is that insideness would be supposed to be cheaper than connectedness. Note that considered selections are organized into trees, which is certainly not a bad way of doing things. Are encircling blobs cheaper ? (more compact, emitting a better information vs. noise ratio ?) This is not absolutely certain.

It is neverteless interesting to study how we can adapt color graphs to higraphs (the result being termed, for convenience, "color higraphs").

As an example, let's consider the *Circle* "color higraph". To map the two scales of pigments, this higraph features the partionning notation (a dashed line separating in two halves the *Circle* blob) : hence two blobs in each half (respectively α, β and λ, μ). Except for the mini-tokens which are better located in color graphs, the result we obtain (next figure) is the accurate transposition of figure 19. (The two unlabelled arrows mark the initial blobs in Harel's notation.)

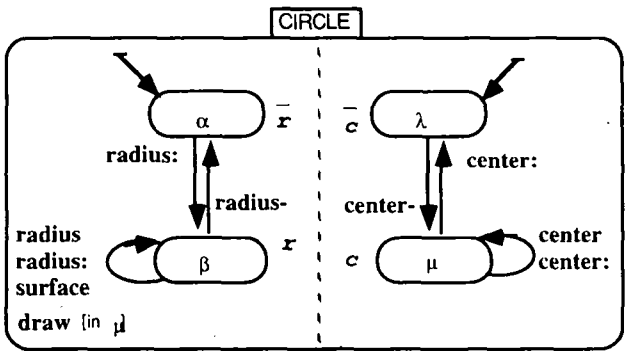
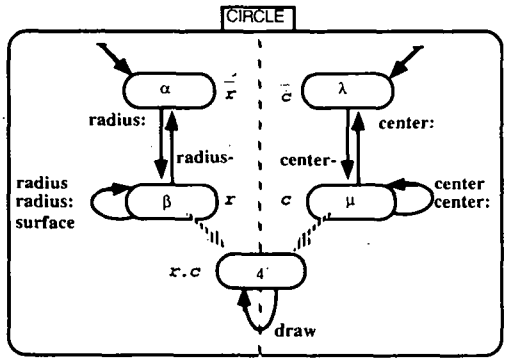
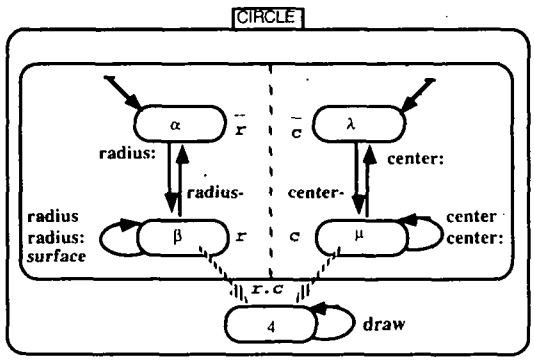


Figure 28.

Note the *draw* transition is represented in the same way in figures 19 and 28 : a circular edge flowing out of β (resp. μ) with a bracketed condition *[in μ]* (resp. *[in β]*). As noticed previously (subsection 4.5.1), this mechanism is basically unsymmetrical. By construction, the statechart standard notation does not offer a mean to materialize the state (β μ) as done in the *Circle* color graph with color 4 (figure 15).

To keep the symmetry, one may draw the equivalent of a multi-micro-transition with two edges, one flowing of out of β and the other one out of μ. But, as explained about color graphs, this is not really satisfying. An alternative solution is to keep the initial connectedness notation in this special occasion. However, this departs from the blob -if not insideness- notation : both possibilities we obtain (see next two figures) require a convention to be adopted.





Figures 29 & 30.

A second alternative consists in partly encircling the two blobs μ and β with a dashed one (see next figure). This last notation is more conform to the blob formalism. However, figures may be difficult to draw in this way : blobs to be partially encircled may be topologically too distant (with other blobs laying in between) due to the number of blobs per partition and/or to a complex partitioning (large number of scales).

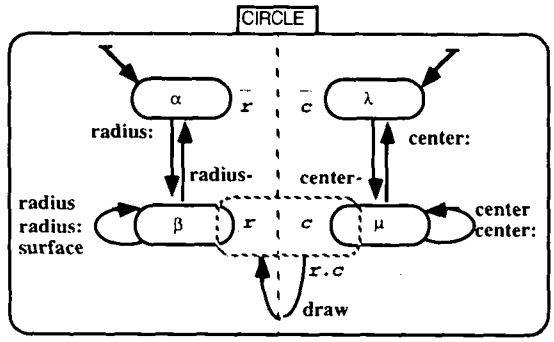


Figure 31.

To keep this dashed blob notation, one way is to draw the intervening blobs on a separate figure (an idea similar to constrained subtrees) : this is modular, but the cognitive load gets increased. (See next figure.) Once again this extension requires a convention to be adopted since it does not match exactly the usual rules of higraphs. An other possibility will be to rely on a 3-dimensions space drawing with animation capability (an a priori more complex solution to live with)...

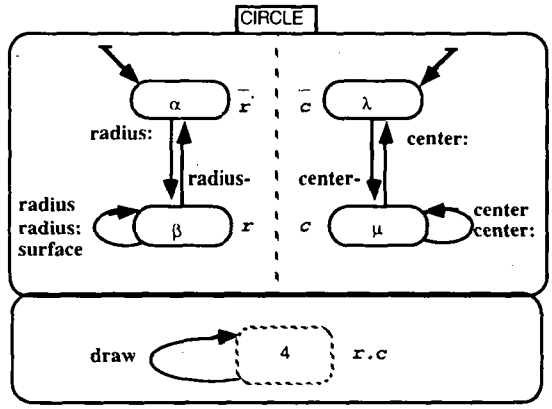


Figure 32.

Higraphs are hierarchical graphs and, as such, they cannot easily be extended to take advantage of blends when these nodes enable more role expressive representations without being too costly. In this respect, the color graph notation (connecteness) appears much more adaptable.

8. CONCLUSION N° 2

In this part, two different points of view have been given for apprehending color graphs.

— Cartesian type representations are clearly unmanageable in general, yet quite useful as far as the concepts are concerned. The color graph formalism offers, from this point of view, a powerfull, yet compact means of specifying states, transitions, methods and memory representations in a N-dimensions space.

— Using insideness in place of connectedness, "color higraphs" are an alternative pragmatic mean to depict the semantics of color graphs. They are almost equivalent to them. Yet, the higraph (more exactly, the statechart) standard notation may not always be easily adapted concerning certain details : as an exercise, we let the reader draw an equivalent of the *Person* color graph (the difficulties stem from the lattice structure due to blends 13 and 14, as well as ephemere pigments 8 and 9). By "explosive effect" [Green, 1982], these difficulties –which can be overcome in small examples with appropriate extensions– may well become serious in real applications. Although not examined here, we also think an advantage of color graphs is to appear probably more "natural" for class inheritance manipulations (ex. : splitting apart the two independent subgraphs of *Person* into a *Sex* class and an *Age* class), and to support a notation more uniform vs. the usual representation of a class lattice. In terms of cost, at least for the applications we are used to (ex. : building, as part of the "Multiworks" ESPRIT Project, a multimedia editor for capturing and elaborating ideas in view of producing possibly complex documents), no decisive argument clearly appears for throwing color graphs overboard. We should certainly be suspicious regarding unforeseen contrary "explosive effects", yet such a decision would be premature commitment. The color graph formalism, a more adaptable tool than its counterpart, is a better support for an experimenation. Being interested by a comparison of genuine color graphs vs. "color higraphs" in a "real world" study, we thus propose to center the planned programming environment around our visual formalism, and to open this one to alternative representations depending on the programmer's task (as a writer or a reader). The help of

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - B.P. 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine - Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 Villers lès Nancy Cedex (France)
Unité de recherche INRIA Rennes - IRISA, Campus Universitaire de Beaulieu 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes - 46, avenue Félix Viallet - 38031 Grenoble Cedex 1 (France)
Unité de recherche INRIA Rocquencourt - Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)

ISSN 0249 - 6399