PASCAL **ROQUES**

UML 2
par la pratique

EYROLLES

 ISBN : 978-2-212-13344-8

Chapitre

6

Modélisation dynamique : exercices corrigés et conseils méthodologiques

Mots-clés

■ Activité continue/finie ■ Transition automatique ■ Événements « `after` » et « `when` » ■ Régions concurrentes ■ Effets d'entrée (`entry`) ou de sortie (`exit`) ■ Action – flot ■ Décision ■ Embranchement – jonction.

Ce chapitre va nous permettre de compléter, au moyen de plusieurs petits exercices, le passage en revue des principales difficultés que pose la construction des diagrammes d'états UML, à savoir :

- activité continue ou finie, transition automatique ;
- pseudo-événements *after* et *when* ;
- régions concurrentes ;
- effets d'entrée (*entry*) et de sortie (*exit*) ;
- points d'entrée et de sortie ;
- héritage de transitions d'un super-état.

Nous reverrons également les bases du diagramme d'activité, ainsi que les nouveautés les plus intéressantes introduites par UML 2.

Nous avons déjà traité des diagrammes de séquence aux chapitres 1 et 2, mais nous les reverrons ici, ainsi que les diagrammes de communication dans la partie consacrée à la conception.

CONCEPTS DE BASE DU DIAGRAMME D'ÉTATS

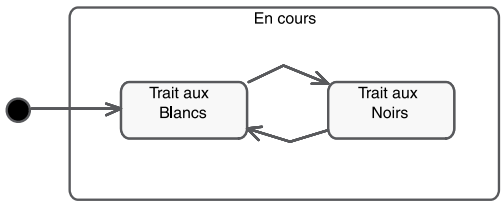
EXERCICE 6-1.
Diagramme d'états d'une partie d'échecs

Dessinez le diagramme d'états correspondant au déroulement d'une partie d'échecs.

solution

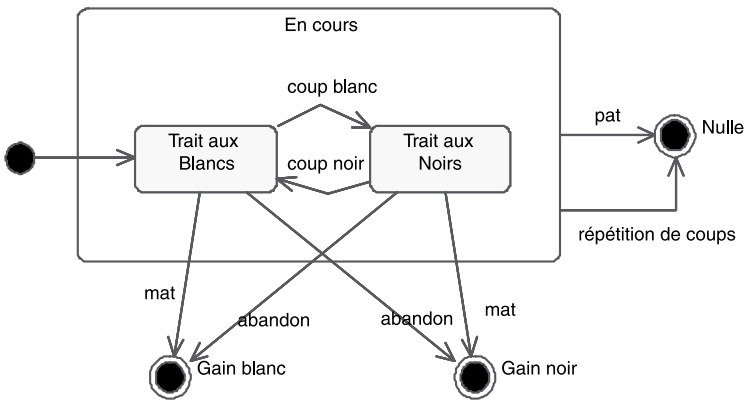
Commençons par représenter le comportement séquentiel d'une partie, sachant que les Blancs commencent.

Figure 6-1.
Début du diagramme d'états de la partie



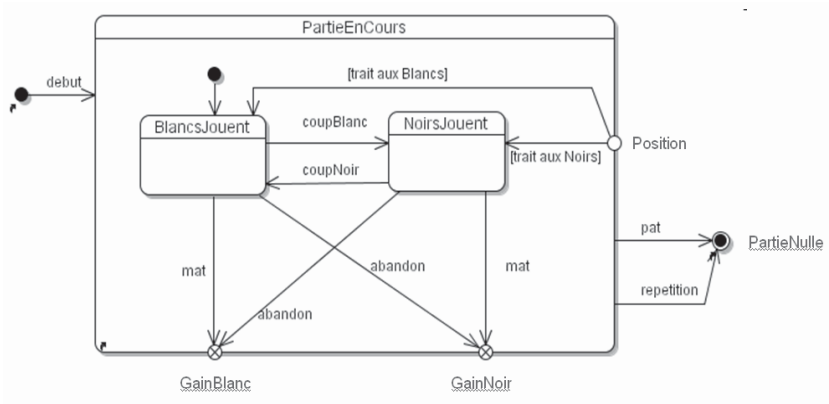
Représentons ensuite par des états finaux différents les trois issues possibles : gain blanc (1-0), g ain noir (0-1) et partie nulle (1/2-1/2). Nous n'a vons pas cherché l'exhaustivité, les règles des échecs de compétition étant nettement plus complexes que ce qui est dessiné sur le schéma suivant.

Figure 6-2.
Diagramme d'états de la partie



Si nous souhaitons ajouter la possibilité de commencer une partie à partir d'une position donnée à la place de la position initiale standard, nous sommes amenés à utiliser la nouv elle notation du point d'entrée (« *entry point* »). Le cercle blanc nommé « Position » sur la figure 6-3 permet de démarrer directement une partie dans l'état « NoirsJouent » si on le désire, alors que le sous-état initial par défaut est « BlancsJouent », comme indiqué par la flèche positionnée au-dessus de ce sous-état. Les événements « pat » et « répétition » sont factorisés, alors que « abandon » et « mat » mènent à des états de sortie différents suivant l'état source. La notation du point de sortie (« *exit point* ») consiste en une croix à l'intérieur d'un cercle blanc. Elle est ég alement nouvelle et propre à UML 2.

Figure 6-3.
Diagramme d'états complété de la partie



EXERCICE 6-2.
Diagramme d'états simple

Considérons un réveille-matin simplifié :

- on peut mettre l'alarme « on » ou « off » ;
- quand l'heure courante de vient ég ale à l'heure d'alarme, le rév eil sonne sans s'arrêter ;
- on peut interrompre la sonnerie.

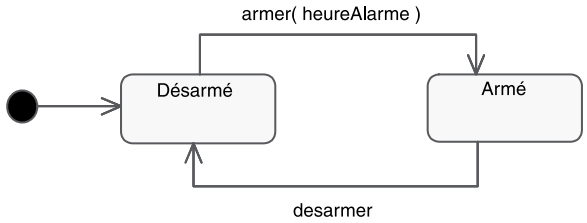
Dessinez le diagramme d'états correspondant.

Solution

Voyons tout d'abord la première phrase :
1. On peut mettre l'alarme « on » ou « off ».

Le réveil a clairement deux états distincts : *Désarmé* (alarme « off ») ou *Armé* (alarme « on »). Une action de l'utilisateur permet de passer d'un état à l'autre. On suppose que le réveil est bien désarmé au départ. Notez le paramètre *heureAlarme* de l'événement *armer*.

Figure 6-4.
Diagramme d'états de la phrase 1

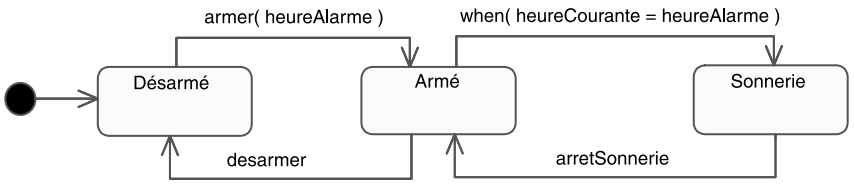


Considérons maintenant les deux autres phrases :

2. *Quand l'heure courante devient égale à l'heure d'alarme, le réveil sonne sans s'arrêter ;*
3. *On peut interrompre la sonnerie.*

Le fait de sonner constitue un nouvel état pour le réveil. Il s'agit bien d'une période de temps durant laquelle le réveil effectue une certaine activité (sonner) qui dure jusqu'à ce qu'un événement vienne l'interrompre.

Figure 6-5.
Diagramme d'états préliminaire du réveille-matin



Le passage de l'état *Armé* à l'état *Sonnerie* est déclenché par une transition due à un changement interne, représenté au moyen du mot-clé « `when` ». En revanche, d'après l'énoncé, le retour de l'état *Sonnerie* à l'état *Armé* ne s'effectue que sur un événement utilisateur.

EXERCICE 6-3.
Activité finie et transition automatique

Complétez le diagramme d'états précédent pour prendre en compte le fait que la sonnerie du réveil s'arrête d'elle-même au bout d'un certain temps.

Solution

Il y a donc une deuxième possibilité de sortie de l'état *Sonnerie* : quand le réveil s'arrête tout seul de sonner au bout d'un certain temps.

À retenir

ACTIVITÉ CONTINUE OU FINIE – TRANSITION AUTOMATIQUE

Une activité durable à l'intérieur d'un état peut être soit :

– « continue » : elle ne cesse que lorsque se produit un événement qui fait sortir l'objet de l'état ;

– « finie » : elle peut également être interrompue par un événement, mais elle cesse de toute façon d'elle-même au bout d'un certain temps, ou quand une certaine condition est remplie.

La transition de complétion d'une activité finie, aussi appelée transition automatique, est représentée en UML sans nom d'événement ni mot-clé.

Dans notre e xemple, il suf fit donc d'ajouter une acti vité durable *sonner* à l'état *Sonnerie* et une transition automatique en sortie de cet état. Le diagramme d'états complété est représenté sur le schéma suivant.

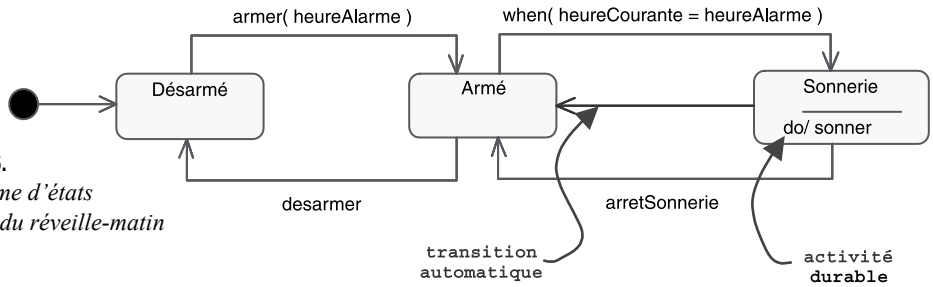


Figure 6-6.
Diagramme d'états complété du réveille-matin

Il convient aussi de se demander si l'utilisateur a le droit de désarmer le réveil pendant qu'il sonne. Dans ce cas, il faudrait ajouter une transition déclenchée par *desarmer* et allant directement de *Sonnerie* à *Désarmé*.

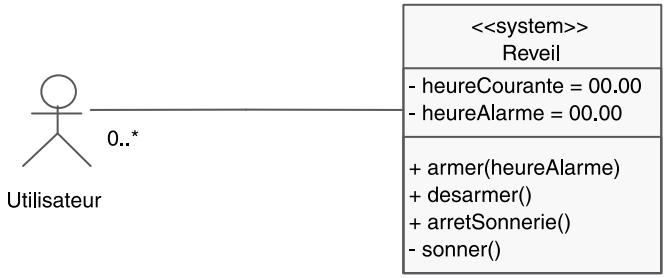
EXERCICE 6-4.
Diagramme de contexte statique

Déduisez-en le diagramme de contexte statique étendu du réveil (voir exercice 5-10).

solution

Si l'on applique de nouveau les règles énoncées lors de l'exercice 5-10, on obtient sans difficulté le diagramme ci-après.

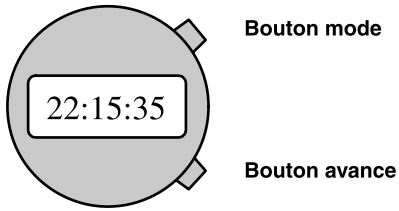
Figure 6-7.
Diagramme de contexte statique étendu



EXERCICE 6-5.
Diagramme d'états simple

Considérons une montre à cadran numérique simplifiée :

Figure 6-8.
Montre à cadran numérique simplifiée



1. Le mode courant est le mode « Affichage ».
2. Quand on appuie une fois sur le bouton mode, la montre passe en « modification heure ». Chaque pression sur le bouton avance incrémente l'heure d'une unité.
3. Quand on appuie une nouvelle fois sur le bouton mode, la montre passe en « modification minute ». Chaque pression sur le bouton avance incrémente les minutes d'une unité.
4. Quand on appuie une nouvelle fois sur le bouton mode, la montre repasse en mode « Affichage ».

Dessinez le diagramme d'états correspondant.

solution

On obtient sans difficulté particulière ce diagramme d'états typique, qui est présenté sur le schéma suivant.

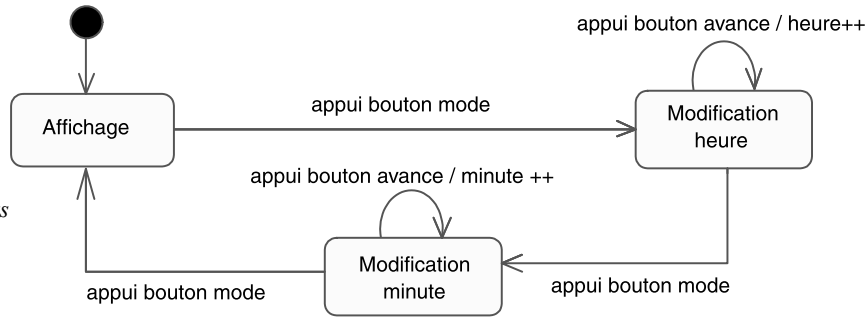


Figure 6-9. *Diagramme d'états préliminaire de la montre à cadran numérique*

On remarquera les notations en style C++ ou Java pour les actions : « heure++ » et « minute++ ». UML n'impose pas de « langage d'action » nous pouvons donc en exprimer le détail comme nous le souhaitons : texte libre, pseudo-code, etc.

Nous obtenons des transitions propres sur les états de modification et pas sur l'état d'affichage. Cela veut-il dire que l'événement « appui bouton avance » est impossible dans l'état « Affichage » ? Non, bien sûr. Cela signifie plutôt que, comme cet événement n'a aucun effet dans cet état, il ne déclenche aucune transition. L'événement est purement et simplement perdu.

CONCEPTS AVANCÉS DU DIAGRAMME D'ÉTATS

EXERCICE 6-6. Événement temporel

Ajoutez le comportement suivant : quand on appuie sur le bouton avance plus de deux secondes, les heures (ou les minutes) s'incrémentent rapidement jusqu'à ce qu'il se produise un relâchement dans la pression du bouton.

Envisagez plusieurs solutions possibles.

solution

Dans l'exemple précédent, les événements d'appui sur les boutons correspondaient en fait au couple indivisible « pression » et « relâchement ». Nous avions considéré que la durée de pression sur chaque bouton était négligeable

par rapport aux durées des états ou, en tout cas, non significati ve. Avec le nouvel énoncé, ce n'est plus le cas, puisque la durée de pression sur le bouton avance influe sur le comportement de la montre. La bonne approche consiste à introduire un nouv el événement : « relâchement bouton a vance », afin de pouvoir gérer le temps de pression.

Le diagramme sui vant montre l'utilisation du nouv eau *timing dia gram* d'UML 2.

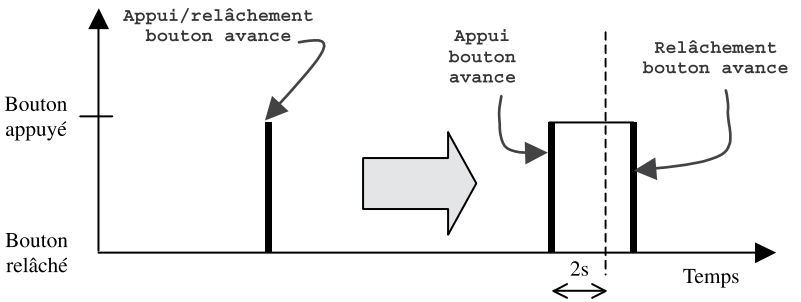


Figure 6-10.
Timing diagram : transformation d'un événement en deux

Une première solution, tentante, consiste à introduire une condition sur la durée de pression, ainsi qu'un nouv el état « Incrémentation rapide », comme cela est illustré sur la figure suivante :

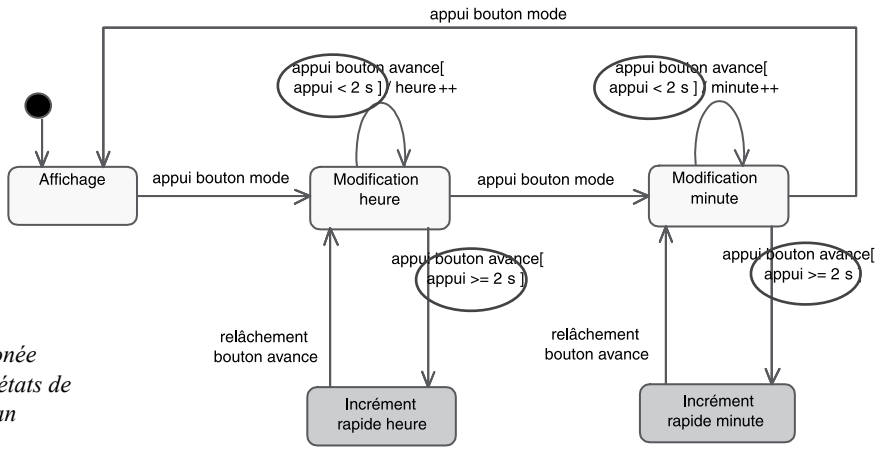


Figure 6-11.
Modification erronée du diagramme d'états de la montre à cadran numérique

Pourtant, cette solution qui semble évidente n'est pas acceptable en UML.

En effet, un événement (comme une transition) est par con vention instantané, ou en tout cas insécable (atomique). Il est donc tout à fait inapproprié de tester

sa durée ! Les seuls concepts dynamiques en UML pour lesquels la notion de durée est significative sont l'état et l'activité durable. Il faut donc s'en servir pour résoudre cet exercice. Deux solutions sont possibles : toutes les deux nécessitent que l'on ajoute un état intermédiaire pour que l'on puisse tester la durée de pression sur le bouton avance, mais elles diffèrent quant à la façon de réaliser ce test :

- La première approche consiste à introduire une activité finie « attendre 2 s » dans l'état intermédiaire et une transition automatique qui représente le fait que le bouton est appuyé plus de deux secondes.
- La seconde approche consiste à utiliser un autre mot-clé proposé par UML : le pseudo-événement « `after` », suivi d'une durée en argument représentant l'échéance d'un timer.

Pour illustrer les deux solutions, nous les avons représentées ensemble sur le diagramme suivant mais, dans la réalité, il faudrait bien sûr en choisir une seule et l'appliquer aux deux états de modification. Pour notre part, nous recommandons la seconde solution qui nous semble plus simple et plus facile à lire.

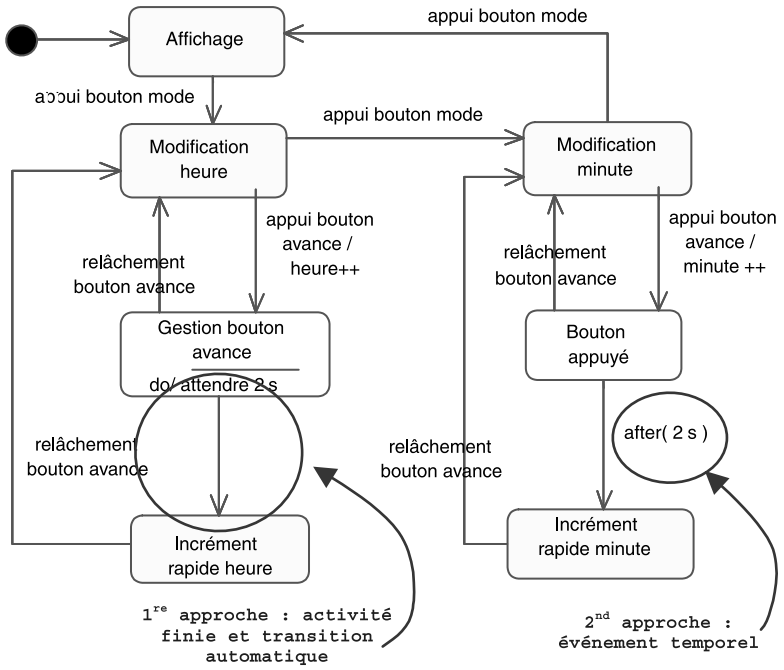


Figure 6-12.
Les deux possibilités pour procéder à une modification correcte du diagramme d'états de la montre à cadran numérique

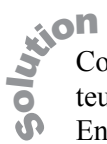
On notera que le comportement initial est conservé : si le bouton avance est relâché en moins de deux secondes, les heures (ou les minutes) sont bien incrémentées d'une unité. En fait, la

transition propre qui existait sur chaque état de modification a pu être coupée en deux suite à la séparation des deux événements « appui » et « relâche », et à l'ajout de l'état intermédiaire.

EXERCICE 6-7.
Récapitulatif

Modélisez le comportement du vidéoprojecteur lors d'une session de formation.

Commencez par identifier les états et transitions « nominaux ». Ajoutez les périodes de préchauffage et de refroidissement de la lampe. Représentez ensuite le fait qu'il faut appuyer successivement deux fois en moins de 5 s sur le bouton power pour interrompre la vidéoprojection. Envisagez enfin la panne de la lampe...



Commençons par identifier le scénario nominal d'utilisation du vidéoprojecteur : le brancher, puis l'allumer (bouton power), puis connecter l'ordinateur. Ensuite, l'éteindre, puis le débrancher.

Si nous ajoutons la possibilité de l'éteindre alors qu'il est allumé ou connecté, puis celle de le débrancher intempestivement, nous arrivons au diagramme de la figure 6-13.

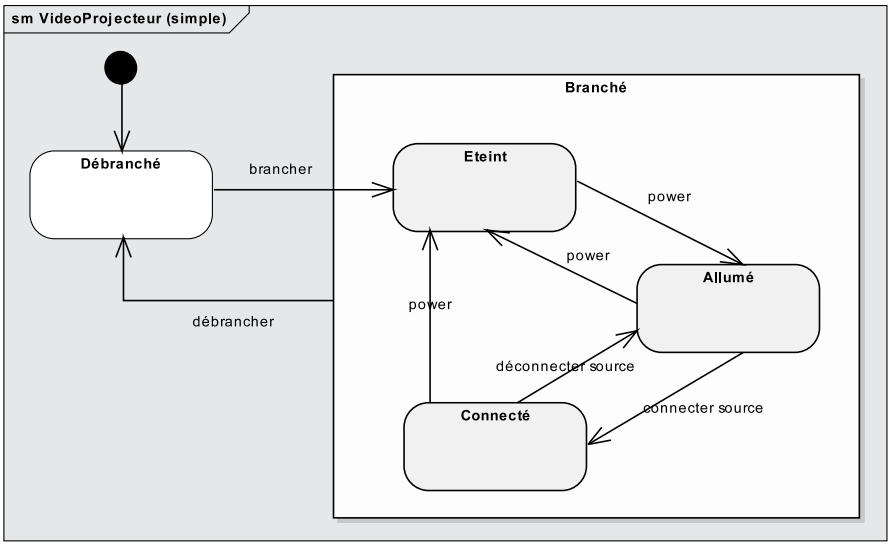


Figure 6-13. *Première version du diagramme d'états du vidéoprojecteur*

Ajoutons le comportement sui vant : si l'on éteint le vidéoprojecteur sans le déconnecter de sa source, il repassera directement dans l'état *Connecté* quand

on le rallumera. Les deux états *Allumé* et *Connecté* intègrent chacun une activité durable : respecti vement la projection d'un écran bleu ou de la source d'entrée. Le diagramme devient alors comme indiqué sur la figure 6-14.

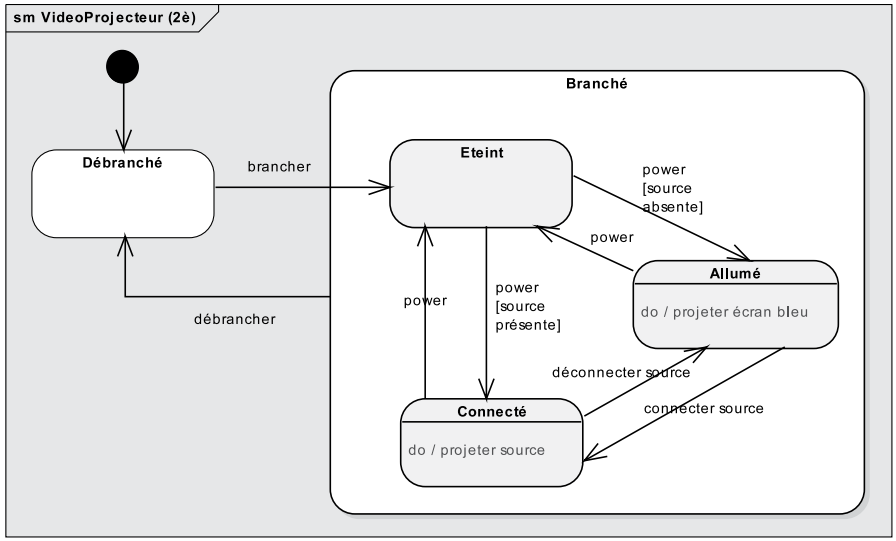


Figure 6-14. *Deuxième version du diagramme d'états du vidéoprojecteur*

Ajoutons maintenant les activités continues de préchauffage et de refroidissement de la lampe. Il est donc nécessaire d'introduire deux états supplémentaires autour de l'état *Éteint*.

Comment sortir de ces états supplémentaires : soit en utilisant un événement de changement (`when`) testant la température de la lampe, soit plus simplement une transition automatique. Nous utiliserons cette dernière solution, plus simple et n'obligeant pas à spécifier les températures visées.

Vérifions que nous avons pris en compte tous les scénarios : d'abord le comportement nominal en début de session de formation : *Débranché* – brancher – *Éteint* – po wer – *Préchauffage* – [source absente] – *Allumé* – connecter source – *Connecté*. Puis, à la pause, le formateur éteint le projecteur : *Connecté* – power – *Refroidissement* – *Éteint*. Ensuite, de retour dans la salle, il rallume le projecteur et n'a pas besoin de reconnecter la source : *Éteint* – power – *Préchauffage* – [source présente] – *Connecté*.

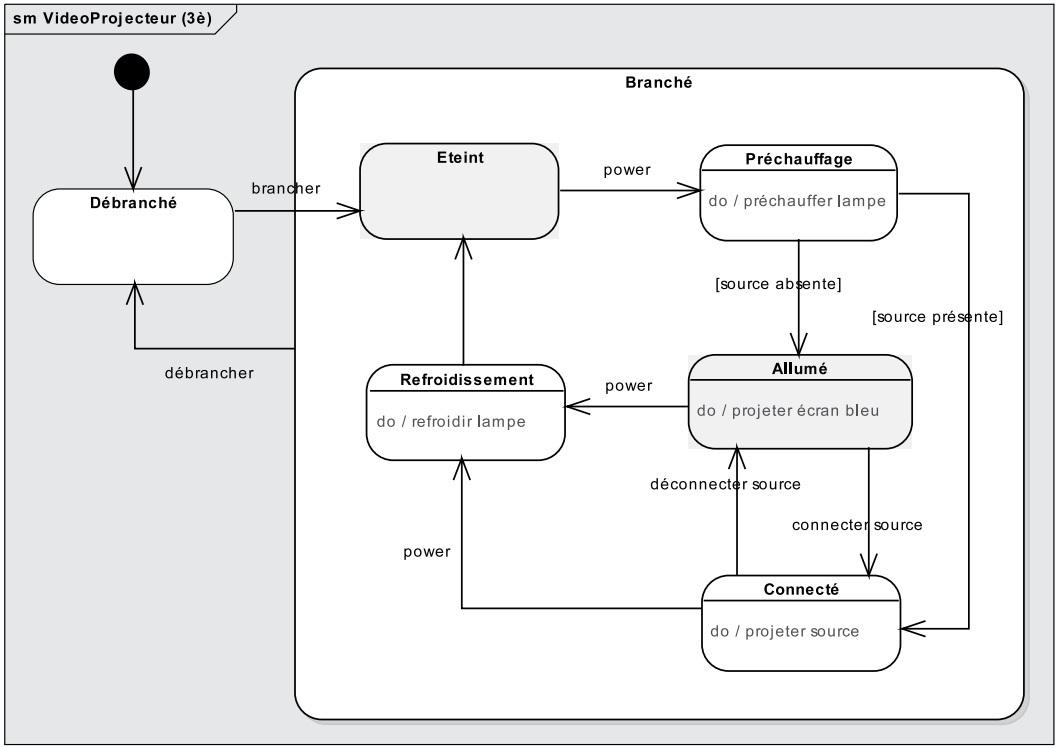


Figure 6-15.
Troisième version du diagramme d'états du vidéoprojecteur

Pour éviter de perdre trop de temps lors d'un appui intempestif sur power dans l'état *Connecté*, les vidéoprojecteurs modernes demandent une confirmation sous la forme d'un deuxième appui sur power en moins de 5 s.

Nous avons vu lors de l'exercice précédent l'événement temporel `after(délai)` qui va nous servir ici, associé à un nouvel état transitoire d'attente de confirmation.

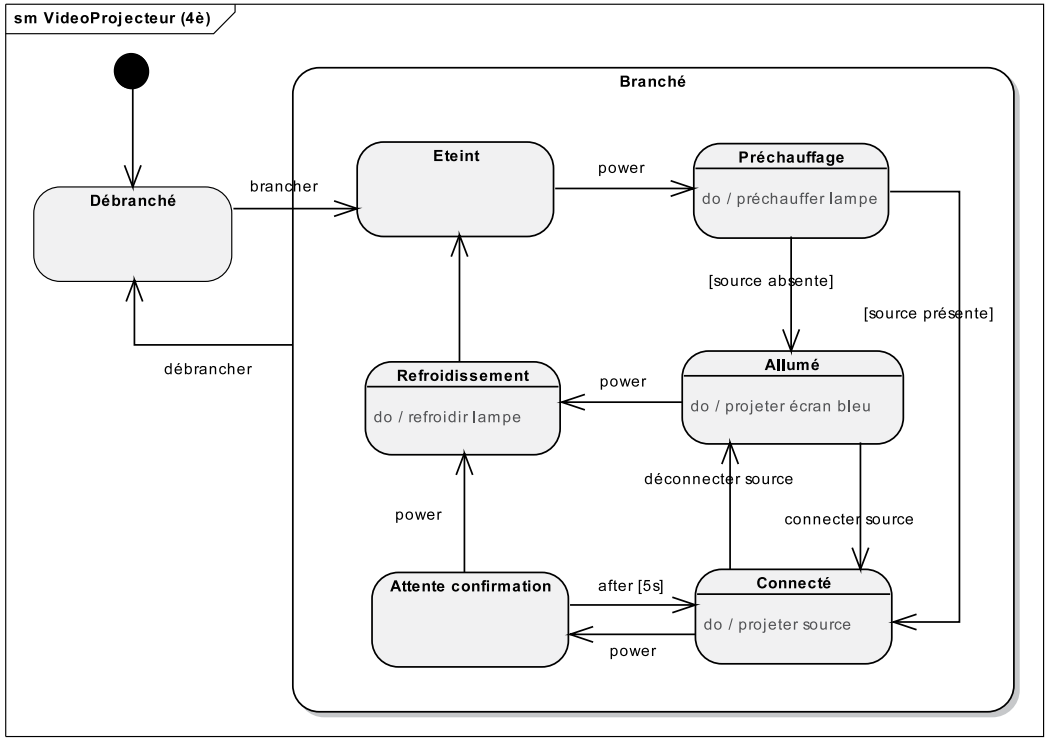


Figure 6-16.
Quatrième version du diagramme d'états du vidéoprojecteur

Ajoutons enfin l'événement redouté : la lampe peut griller dès lors que le projecteur n'est pas éteint. Le plus simple consiste à introduire un nouv el état composite à l'intérieur de *Branché* mais excluant *Éteint*. Il suffit alors d'introduire une transition factorisée déclenchée par l'événement interne de changement `when` (`état lampe = grillée`), qui amène vers un état de panne. Le diagramme complété est représenté sur la figure 6-17.

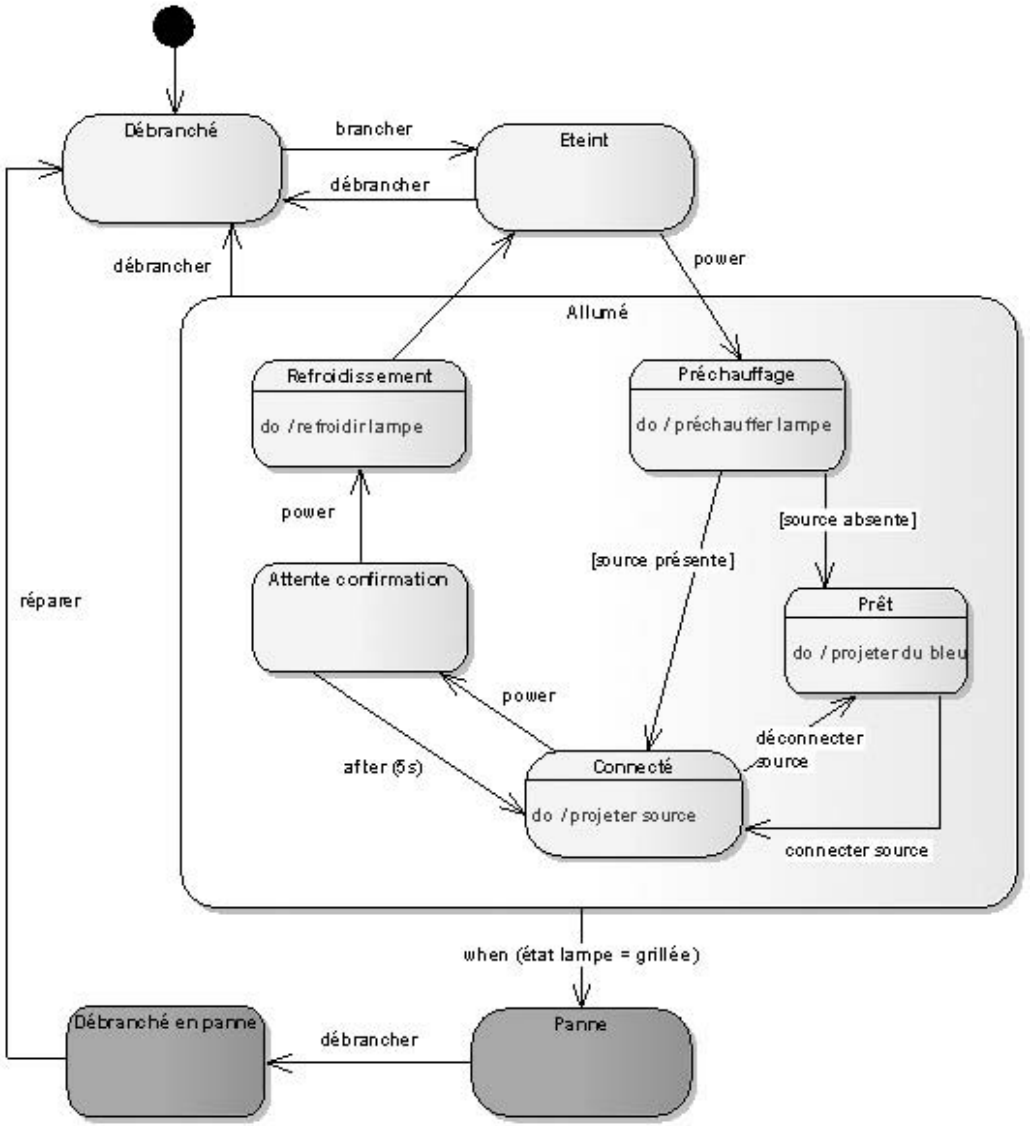


Figure 6-17.
Version du diagramme d'états du vidéoprojecteur avec états de panne

En fait, la confirmation pour éteindre le vidéoprojecteur est également obligatoire depuis l'état *Prêt*. On peut donc introduire un super -état englobant *Prêt* et *Connecté*, mais il faut alors utiliser un pseudo-état *History* (H) pour sa voir dans quel sous-état re venir quand la temporisation tombe. Le schéma finalisé est montré sur la figure suivante.

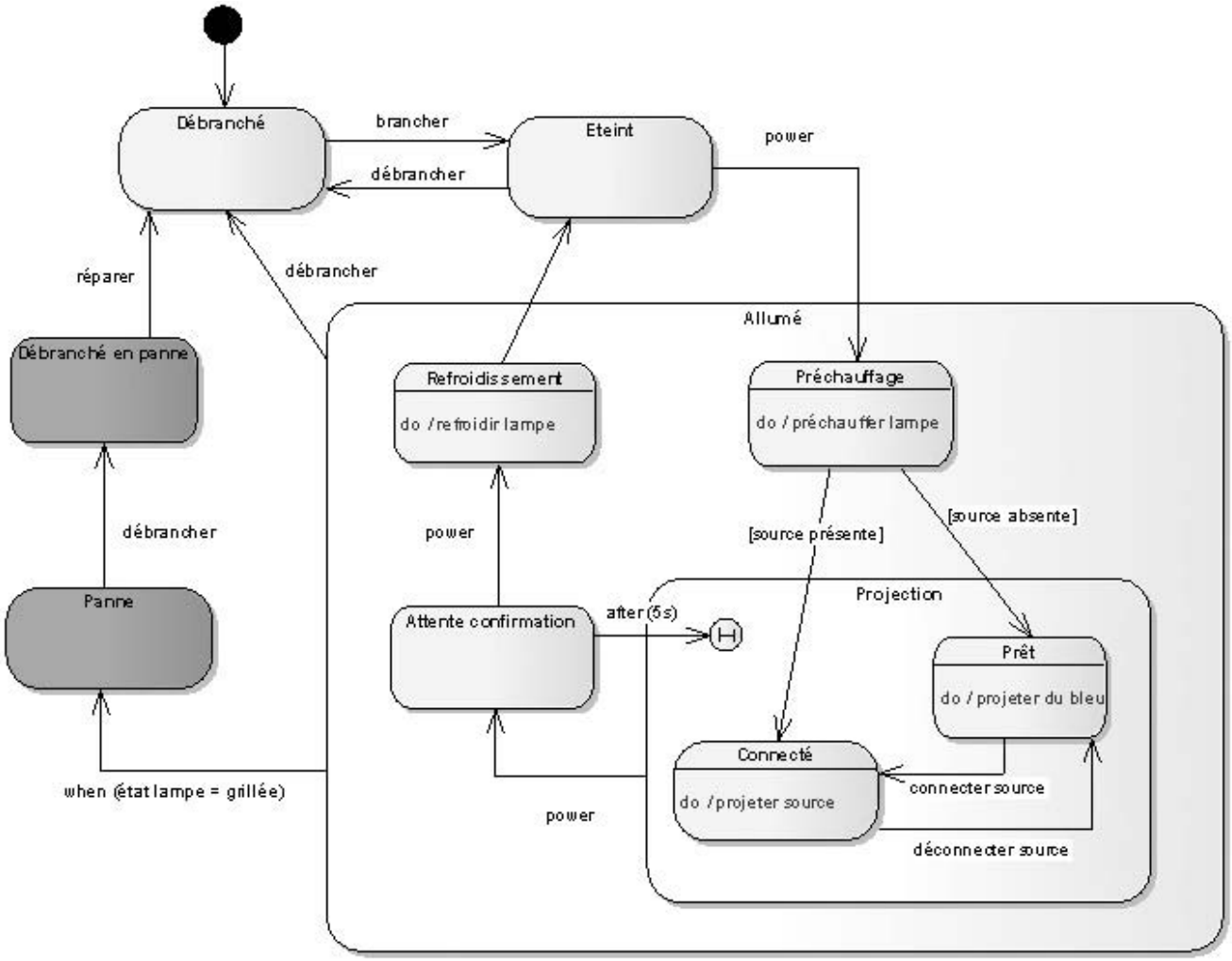
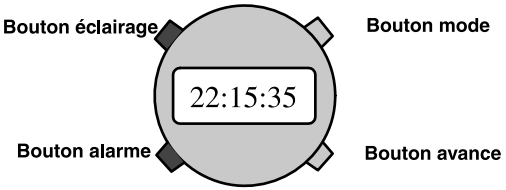


Figure 6-18.
Version du diagramme d'états du vidéoprojecteur

EXERCICE 6-8.
Régions concurrentes

Reprenons notre e xemple de montre à cadran numérique tel qu'il était présenté au début de l'exercice, et ajoutons maintenant à cette dernière deux autres boutons :



- un bouton éclairage ; en le pressant, on éclaire le cadran de la montre, jusqu'à ce qu'on le relâche ;
- un bouton alarme, qui ajoute à la montre digitale une fonctionnalité classique d'alarme, comme cela a été décrit lors du premier exercice de ce chapitre (réveille-matin).

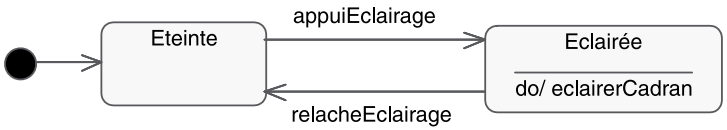
Dessinez le diagramme d'états complet incluant tous les comportements de la montre.

Solution

Nous sommes clairement en présence de trois comportements concurrents :
- la gestion de l'affichage ;
- la gestion de l'alarme ;
- la gestion de l'éclairage.

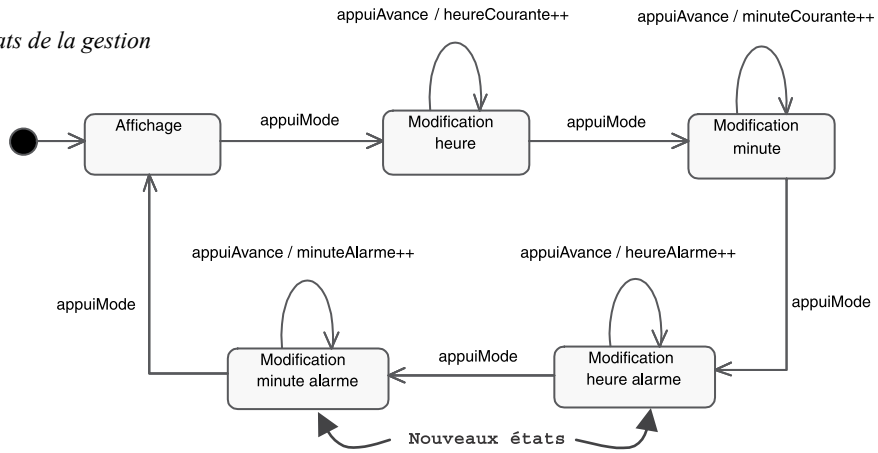
Commençons par le plus simple d'entre eux, qui concerne la gestion de l'éclairage. Cette dernière peut se modéliser très simplement par un automate à deux états, comme nous l'illustrons sur le schéma suivant.

Figure 6-19.
Diagramme d'états de la gestion de l'éclairage



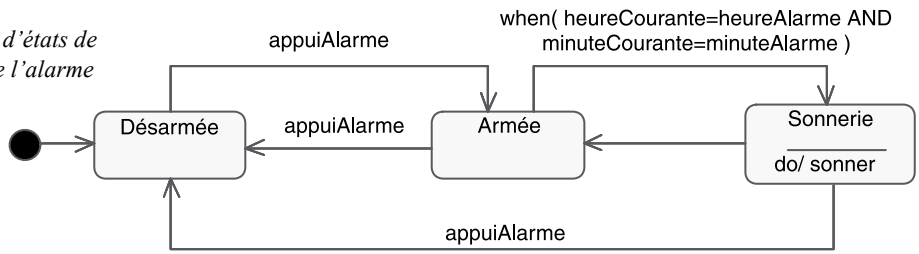
Si la gestion de l'éclairage peut tout à f ait se modéliser à part, il n'en est pas de même pour l'af fichage et l'alarme. En ef fet, il f aut maintenant pouv oir aussi modifier l'heure d'alarme et la minute d'alarme, ce qui ajoute deux nouveaux états au diagramme de la figure 6-9, comme cela est indiqué ci-après.

Figure 6-20.
Diagramme d'états de la gestion de l'affichage



Il reste maintenant à modéliser la gestion de l'alarme. Nous pouvons nous inspirer du diagramme d'états du réveil (voir figure 6-6) pour obtenir le schéma suivant. On y notera la dépendance avec la gestion de l'affichage *via* le test effectué par la gestion de l'alarme sur les attributs (« `when` »…).

Figure 6-21.
Diagramme d'états de la gestion de l'alarme



Nous avons donc obtenu trois diagrammes d'états. Comment faire en sorte que ces trois diagrammes séparés décrivent le comportement de la montre à cadran numérique ?

Là encore, deux solutions sont possibles :

- Considérer que toute instance de *Montre* contient en fait trois instances et que chacune gère un des trois comportements décrits précédemment. Toute montre délègue ainsi une partie de sa dynamique à une instance d'afficheur, d'éclairage ou d'alarme, suivant le cas. On peut représenter cela au moyen d'une relation de composition dans un diagramme de classes. Toutefois, UML 2 permet une meilleure solution : utiliser le concept de classe structurée (revoir l'exercice 4-5), ou même de composant. Nous en profitons pour illustrer le connecteur de délégation (« dele gate »). Celui-ci relie le

contrat externe du composant (ses interfaces) à la réalisation de ce comportement par ses parties.

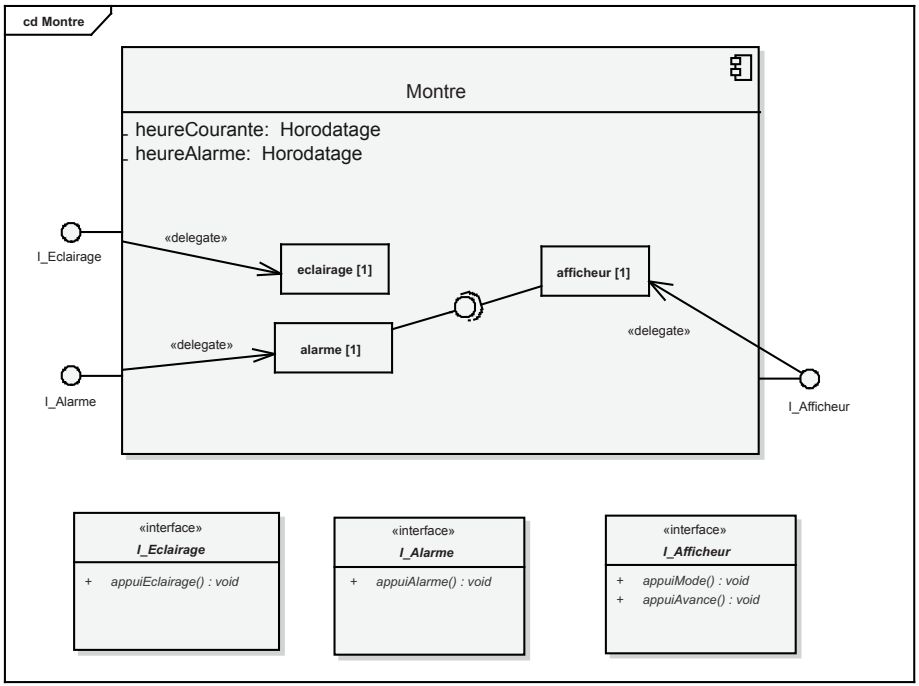
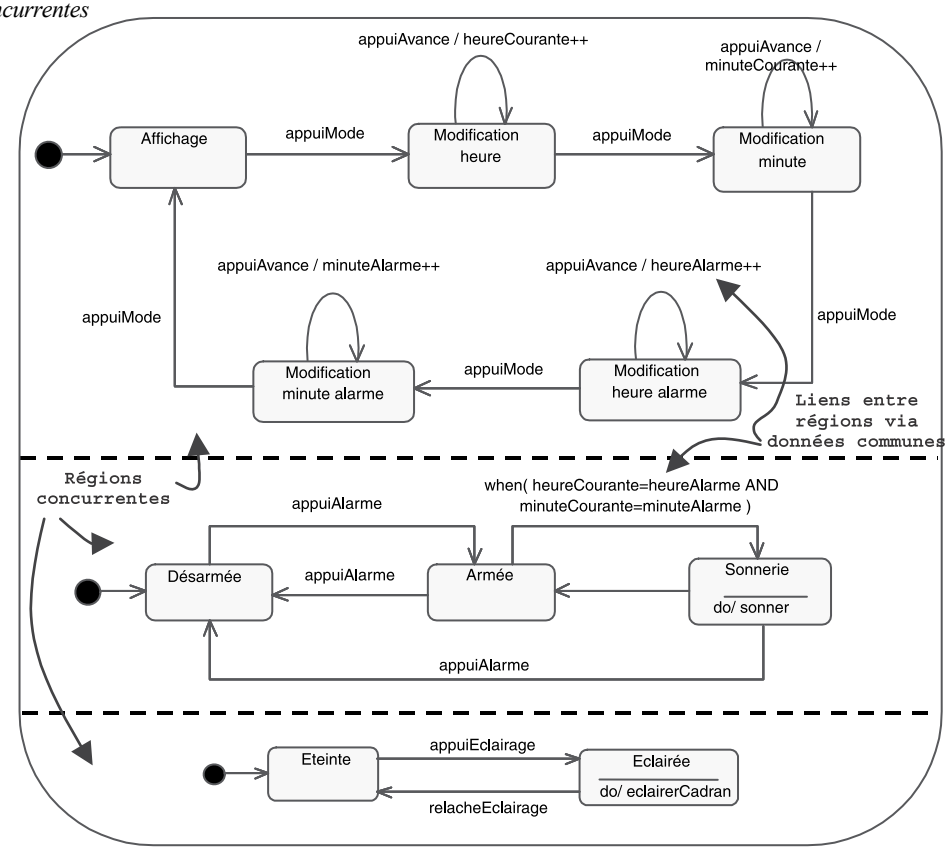


Figure 6-22. *Diagramme de structure composite qui montre la relation de délégation*

- Décrire des « régions concurrentes » au sein du diagramme d'états de la classe *Montre*. L'état courant de la montre de vient alors un v ecteur à trois lignes : état de l'affichage, état de l'alarme, état de l'éclairage. Une montre peut simultanément être en af fichage de modification minute, être en train de sonner et être éclairée.

Le diagramme d'états de la montre a vec trois régions concurrentes apparaît sur le schéma ci-après.

Figure 6-23.
Diagramme d'états de la montre avec régions concurrentes

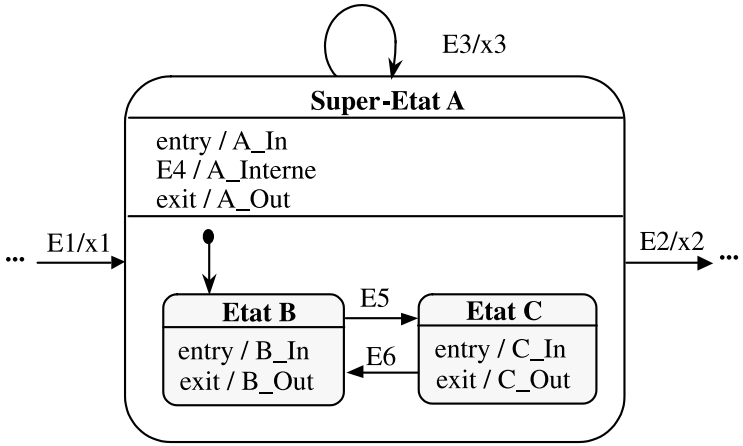


On notera que chaque « région » doit être initialisée puisque, si les états sont bien exclusifs à l'intérieur d'une région concurrente, ils existent simultanément dans les trois régions.

EXERCICE 6-9.
Diagramme d'états hiérarchique

Considérons le fragment de diagramme d'états suivant, qui contient de nombreux effets.

Figure 6-24.
Exemple de diagramme d'états complexe



À retenir

EFFETS D'ENTRÉE (OU DE SORTIE) – ENTRY (EXIT)

Un effet d'entrée (introduit par le mot-clé « `entry` » à l'intérieur du symbole d'un état) représente une action ou une activité qui est exécutée chaque fois que l'on entre dans cet état. Cela permet de factoriser un même effet qui sera déclenché par toutes les transitions qui entrent dans l'état.

L'effet de sortie (introduit par le mot-clé « `exit` ») est l'effet symétrique en sortie de l'état.

Le diagramme de l'énoncé comprend donc :

- une transition propre sur le super-état (E3/x3) ;
- une transition interne dans le super-état (E4/A_Interne) ;
- des transitions d'entrée et de sortie dans le super-état et chacun des sous-états.

Nous allons étudier l'ordre temporel d'exécution des effets en complétant le tableau suivant. Nous partirons de l'état à gauche du diagramme symbolisé par « … », et nous prendrons comme nouvel état de départ l'état d'arrivée de la ligne précédente.

État de départ	Événement	Effets	État d'arrivée
…	E1	?	?
?	E5	?	?
?	E4	?	?
?	E6	?	?
?	E3	?	?
?	E5	?	?
?	E3	?	?
?	E2	?	?

Complétez le tableau précédent.

Solution

Dans l'état de départ, symbolisé par « … » à g auche du diagramme, l'événement E1 déclenche l'effet x1, puis conduit au super-état A. Cette entrée dans le super-état A déclenche l'effet d'entrée A_In, puis l'entrée dans le sous-état B (à cause du symbole du sous-état initial), et donc l'effet d'entrée B_In.

État de départ	Événement	Effets	État d'arrivée
…	E1	x1, A_In, B_In	B (dans A)

Dans l'état B, l'événement E5 f ait sortir de l'état et déclenche donc l'ef fet B_Out, puis conduit à l'état C et déclenche en conséquence C_In.

État de départ	Événement	Effets	État d'arrivée
B	E5	B_Out, C_In	C (dans A)

Dans l'état C, l'événement E4 est-il possible ? Oui, car les transitions inter - nes sont héritées du super-état. L'événement E4 fait donc rester dans l'état C et déclenche simplement l'effet A_Interne.

État de départ	Événement	Effets	État d'arrivée
C	E4	A_Interne	C (dans A)

Dans l'état C, l'événement E6 fait sortir de l'état et déclenche donc C_Out, puis conduit à l'état B et déclenche en conséquence B_In.

État de départ	Événement	Effets	État d'arrivée
C	E6	C_Out, B_In	B (dans A)

Dans l'état B, l'événement E3 est-il possible ? Oui, car les transitions propres sont héritées du super-état. L'événement E3 fait d'abord sortir de l'état B et déclenche l'effet B_Out, puis fait sortir du super-état A et déclenche A_Out, déclenche ensuite x3, puis fait entrer dans le super-état A et déclenche A_In, enfin fait rentrer dans l'état B et déclenche B_In.

État de départ	Événement	Effets	État d'arrivée
B	E3	B_Out, A_Out, x3, A_In, B_In	B (dans A)

Nous avons déjà considéré l'arrivée de E5 dans l'état B :

État de départ	Événement	Effets	État d'arrivée
B	E5	B_Out, C_In	C (dans A)

Attention, il y a un piège ! Dans l'état C, l'événement E3 fait d'abord sortir de l'état C et déclenche C_Out, puis fait sortir du super-état A et déclenche A_Out, déclenche ensuite x3, puis fait entrer dans le super-état A et déclenche A_In, enfin fait rentrer dans l'état B (car c'est le sous-état initial !) et déclenche B_In.

État de départ	Événement	Effets	État d'arrivée
C	E3	C_Out, A_Out, x3, A_In, B_In	B (dans A)

Dans l'état B, l'événement E2 fait d'abord sortir de l'état B et déclenche B_Out, puis du super-état A et déclenche A_Out, et déclenche enfin x2.

État de départ	Événement	Effets	État d'arrivée
B	E2	B_Out, A_Out, x2	…

CONCEPTS DE BASE DU DIAGRAMME D'ACTIVITÉ

EXERCICE 6-10.
Recette de cuisine !

Recette simplifiée : commencer par casser le chocolat en morceaux, puis le faire fondre.

En parallèle, casser les œufs en séparant les blancs des jaunes.

Quand le chocolat est fondu, ajouter les jaunes d'œuf.

Battre les blancs en neige jusqu'à ce qu'ils soient bien fermes.

Les incorporer délicatement à la préparation chocolat sans les briser.

Verser dans des ramequins individuels.

Mettre au frais au moins 3 heures au réfrigérateur avant de servir.

Représentez par un diagramme d'activité la recette de la mousse au chocolat...

Proposez d'abord une version simple, en supposant que vous avez des ressources illimitées, puis une version avec deux personnes. Complétez ensuite le diagramme en ajoutant soit des flots d'objets soit des *input* et *output pins*.

Solution

Le diagramme d'activité simple est donné par la figure suivante. Nous supposons avoir des ressources illimitées, donc nous parallélisons au maximum.

Notez l'utilisation des barres d'embranchement (*fork*) et de jointure (*join*), permettant de représenter précisément le parallélisme dans les flots de contrôle de l'activité.

Notez ég alement l'utilisation du symbole graphique UML 2 pour l'action « Attendre 3h » qui est de type : *accept time event*.

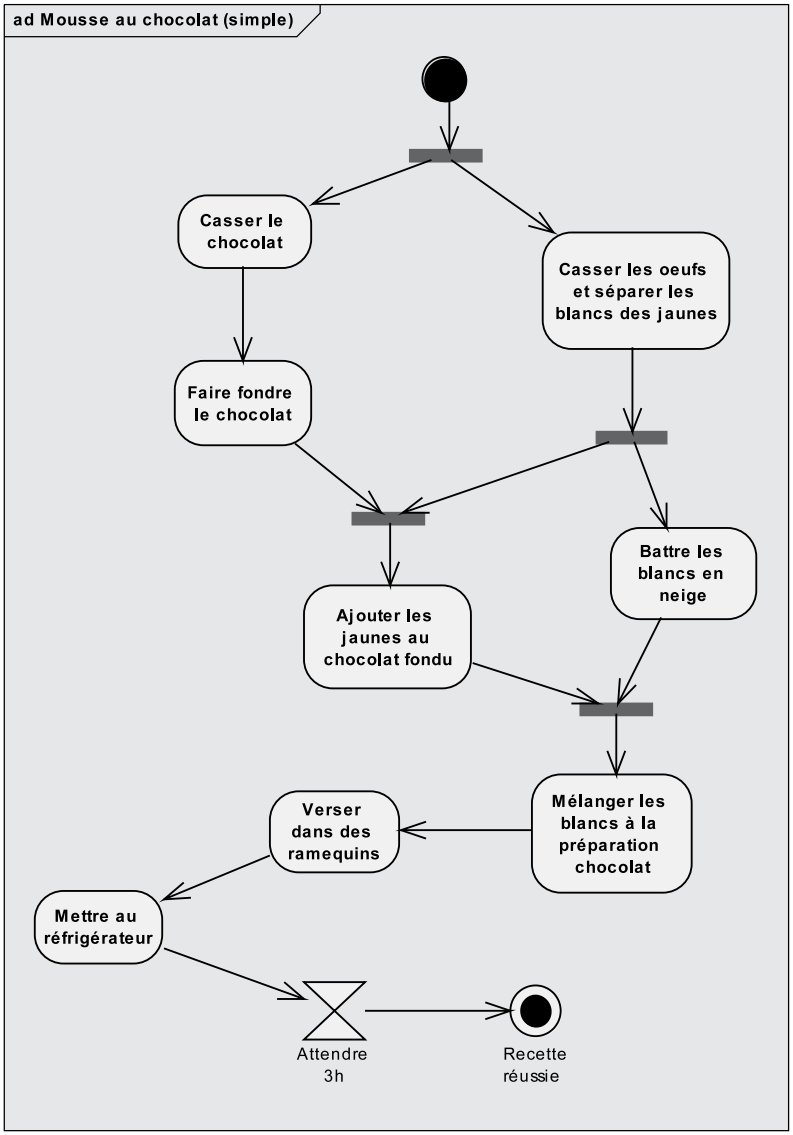


Figure 6-25.
Exemple de diagramme d'activité simple

Si nous supposons maintenant que nous avons deux ressources, un chef et un apprenti, nous pouvons répartir les actions dans deux partitions.

À retenir

PARTITION

Les actions d'un diagramme d'activité peuvent être réparties dans des partitions séparées représentant les entités responsables de ces actions. Il s'agit d'une généralisation du concept UML 1 de « swimlane ».

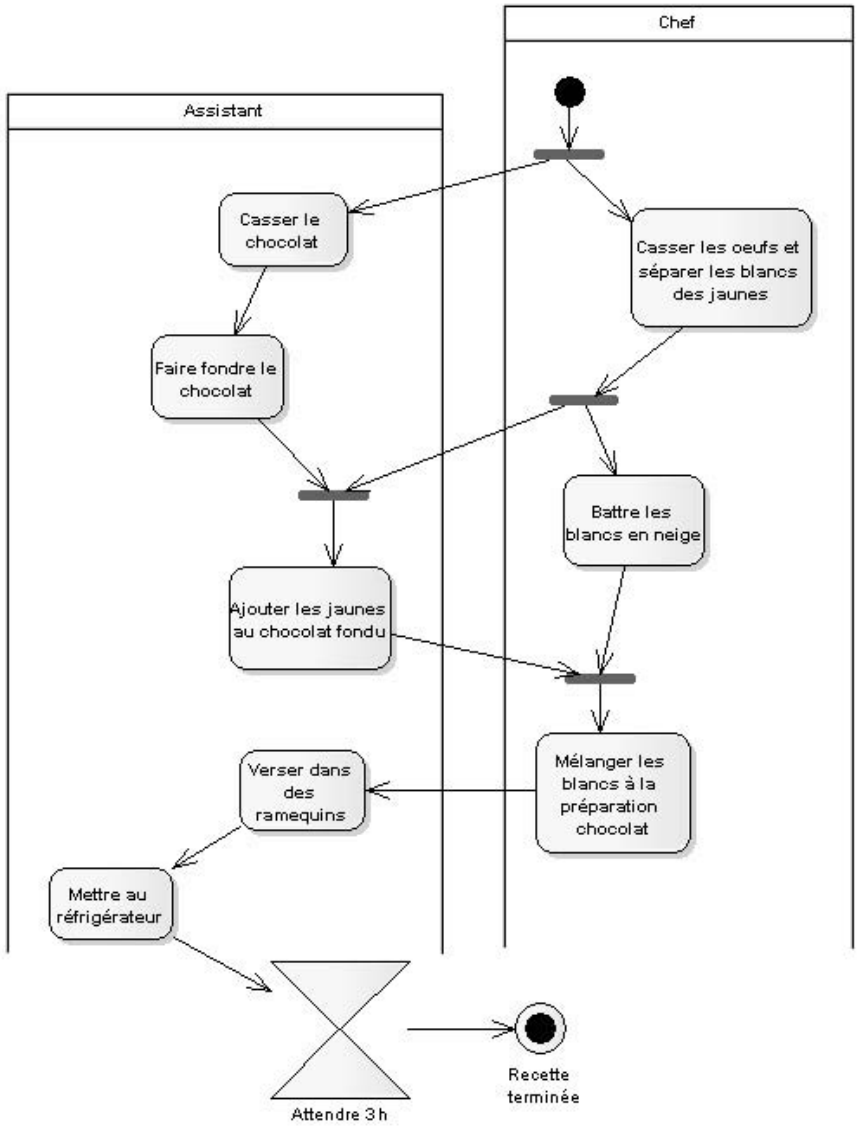


Figure 6-26.
Exemple de diagramme d'activité avec partitions

Nous allons maintenant compléter ce diagramme en ajoutant des flots d'objets, comme illustré sur la figure suivante.

À retenir

FLOTS DE CONTRÔLE ET FLOTS D'OBJETS

UML 2 a unifié la notation des flots de contrôle et des flots d'objets.

Les flots de contrôle connectent des actions pour indiquer que l'action pointée par la flèche ne peut pas démarrer tant que l'action source n'est pas terminée.

Les flots d'objets connectent des nœuds d'objets pour fournir des entrées (*inputs*) aux actions. Le nom d'un état peut être mis entre crochets et placé avec le nom du type.

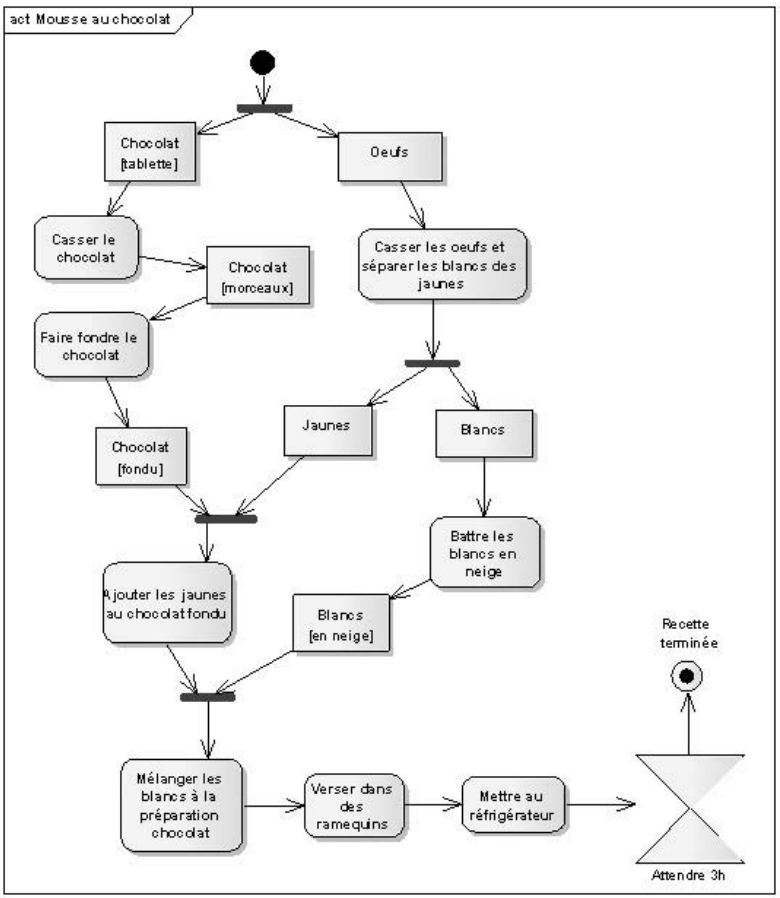


Figure 6-27.
Exemple de diagramme d'activité avec flots d'objets

À retenir

DEUX NOTATIONS POUR LES FLOTS D'OBJETS

UML 2 propose deux notations équivalentes pour les flots d'objets.

Nous avons vu la première sur la figure précédente. Une autre notation possible consiste à indiquer les valeurs d'entrée à la disposition de l'action et celles qu'elle fournit en sortie sous la forme de petits carrés appelés *pins*. Le nom du nœud objet est spécifié à proximité du *pin*.

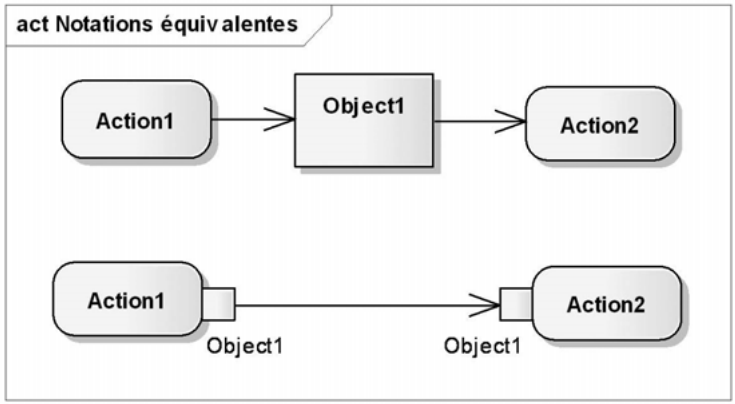


Figure 6-28.
Notations des flots d'objets

Si l'on applique cette deuxième notation à la recette de mousse au chocolat, on aboutit au diagramme de la figure suivante (6-29).

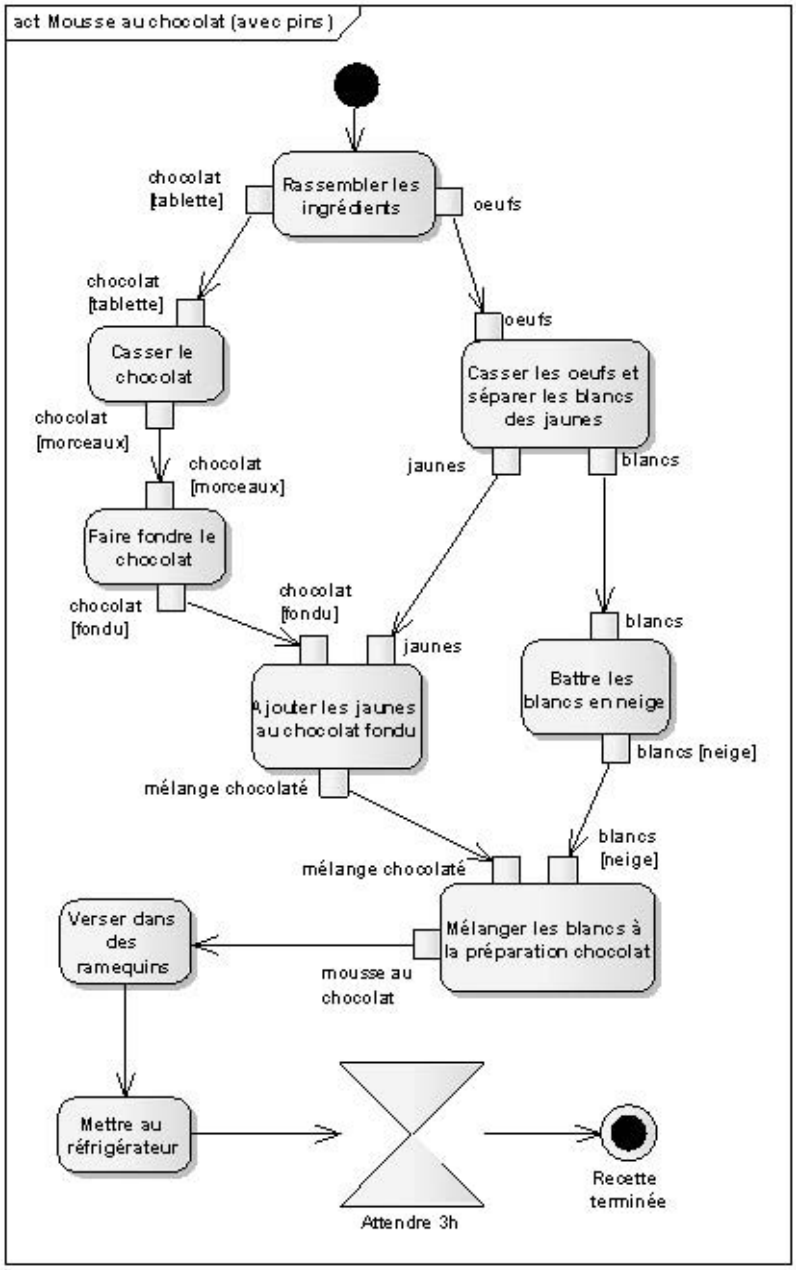


Figure 6-29.
Exemple de diagramme d'activité avec input *et* output pins

CONCEPTS AVANCÉS DU DIAGRAMME D'ACTIVITÉ

EXERCICE 6-11.
Région d'expansion

Précisez le diagramme d'activité précédent en introduisant une région d'expansion pour mieux expliquer la boucle sur le remplissage des ramequins...

À retenir

RÉGION D'EXPANSION

Une région d'e xpansion est un nœud d'acti vité structuré qui s'exécute une fois pour chaque élément dans une collection d'entrée. Les entrées et sorties de la collection (nœuds d'e xpansion) sont matérialisés sur la frontière de la boîte par des petits rectangles divisés en plusieurs compartiments.

Solution

Nous allons utiliser ce nouv eau concept UML 2 pour décrire le remplissage successif des ramequins en fonction de la quantité de mélange chocolat produit. La partie du diagramme modifiée est représentée sur la figure suivante.

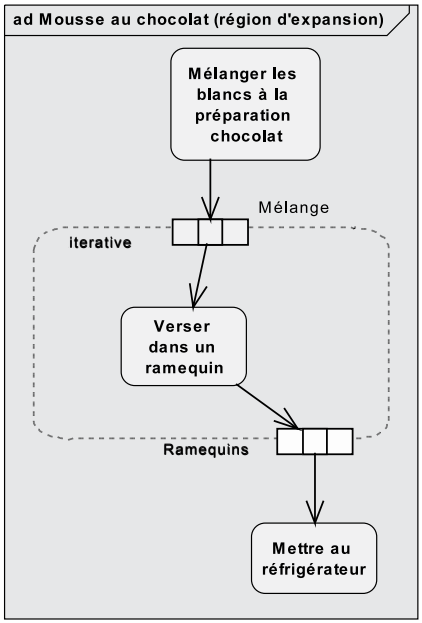


Figure 6-30.
Exemple de région d'expansion

EXERCICE 6-12.
Région d'expansion et flot d'exception

Précisez maintenant le début du diagramme d'activité en introduisant une région d'expansion pour mieux expliquer la séparation des blancs des jaunes ainsi qu'un flot d'exception pour prendre en compte l'oubli du chocolat sur le feu...

solution

Nous avons vu à la question précédente comment utiliser la région d'expansion. Notez également le nouveau symbole UML 2 du nœud final de flot (*flow final*), permettant d'exprimer le fait que si le cuisinier rate la séparation du jaune et du blanc, il ne remplit pas les nœuds d'expansion de sortie.

Si nous voulons ajouter l'exception possible consistant à laisser brûler le chocolat pendant qu'on s'occupe des œufs, il faut introduire le concept de région interruptible et de flot d'exception.

À retenir

RÉGION INTERRUPTIBLE

Une région interruptible est un ensemble de nœuds et d'arcs d'activité au sein duquel l'activité se termine si l'événement désigné se produit. Cet événement d'interruption apparaît comme une flèche « éclair » partant de l'intérieur de la région interruptible et se dirigeant vers la bordure du gestionnaire d'interruption.

La figure suivante illustre l'utilisation de ces nouveaux concepts UML 2 en montrant le diagramme d'activité complet.

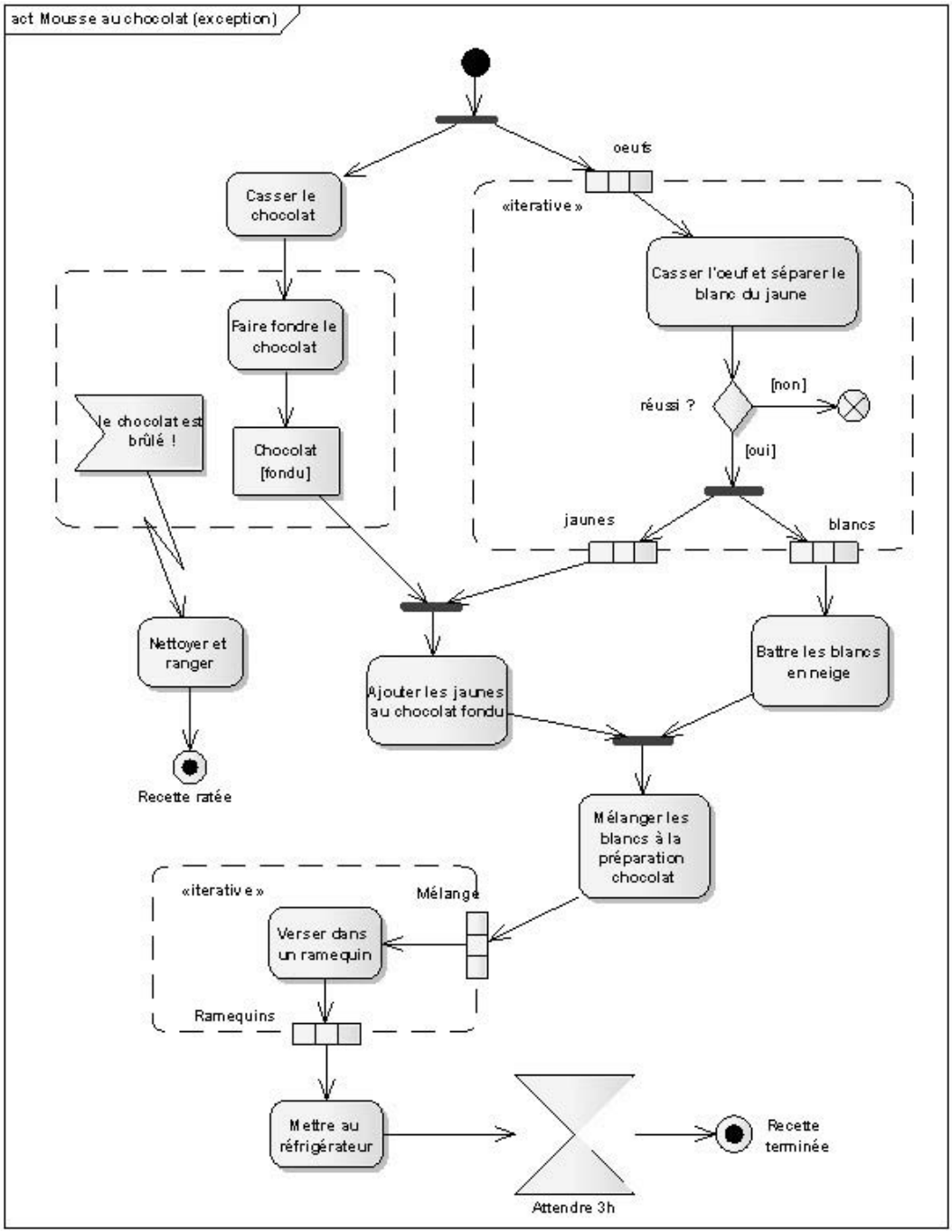


Figure 6-31.
Diagramme d'activité complété de la recette de mousse au chocolat

CONSEILS MÉTHODOLOGIQUES

CONTEXTE DYNAMIQUE

Pour représenter le contexte dynamique, utilisez un diagramme de *communication*, de la façon suivante :

- le système étudié est représenté par un objet au centre du diagramme ;
- cet objet central est entouré par une instance de chaque acteur ;
- un lien relie le système à chacun des acteurs ;
- sur chaque lien sont répertoriés tous les messages en entrée et en sortie du système, sans numérotation.

COMMENT CONSTRUIRE LES DIAGRAMMES D'ÉTATS ?

Pour construire efficacement les diagrammes d'états :

- représentez d'abord la séquence d'états qui décrit le comportement nominal d'une instance, avec les transitions associées ;
- ajoutez progressivement les transitions qui correspondent aux comportements « alternatifs » ou d'exception ;
- complétez les effets sur les transitions et dans les états ;
- structurez le tout en sous-états et utilisez les notations avancées (`entry`, `exit`, etc.) si le diagramme devient trop complexe.

ÉVÉNEMENTS

Distinguez bien les événements internes (« `when`(condition) ») et temporels (« `after`(durée) ») de ceux qui résultent de la réception de messages.

Attention : un événement (comme une transition) est par convention instantané, ou en tout cas insécable (atomique). Il est donc tout à fait incorrect de tester sa durée ! Les seuls concepts dynamiques en UML qui possèdent la notion de durée sont l'état et l'activité durable.

SUPER-ÉTAT

Pensez à utiliser le concept de super-état pour factoriser les nombreuses transitions déclenchées par le même événement et amenant au même état.

EFFET ET CONDITION

Attention, sur une transition, l'effet est toujours déclenché *après* l'évaluation de la condition de garde.

TRANSITION AUTOMATIQUE

Utilisez correctement les transitions automatiques. Une activité durable à l'intérieur d'un état peut être soit :

- « Continue » : elle ne s'arrête que lorsque se produit un événement qui fait sortir de l'état ;
- « Finie » : elle peut également être interrompue par un événement, mais s'arrête de toute façon d'elle-même au bout d'un certain temps, ou quand une certaine condition est remplie.

La transition de complétion d'une activité durable, aussi appelée *transition automatique*, est représentée en UML sans nom d'événement ni mot-clé.

TRANSITION PROPRE OU INTERNE ?

Retenez bien la différence entre la transition propre et la transition interne :

- Dans le cas d'une *transition propre*, l'objet quitte son état de départ pour y revenir ensuite. Cela peut avoir des conséquences secondaires non négligeables comme l'interruption puis le redémarrage d'une activité durable, la réalisation d'effets en entrée (« `entry` ») ou en sortie (« `exit` ») de l'état, etc. ; en outre, si l'état est décomposé en sous-états, une transition propre ramène forcément l'objet dans le sous-état initial.
- En revanche, la *transition interne* représente un couple (événement/effet) qui n'a aucune influence sur l'état courant. La transition interne est notée graphiquement à l'intérieur du symbole de l'état.

PSEUDO-ÉTAT « HISTORY »

Il faut savoir utiliser à bon escient le pseudo-état « `history` » : il permet à un super-état de se souvenir du dernier sous-état séquentiel qui était actif avant une transition sortante. Une transition vers l'état « `history` » rend de nouveau actif le dernier sous-état actif, au lieu de ramener vers le sous-état initial.

EFFETS D'ENTRÉE ET DE SORTIE

N'abusez pas des effets d'entrée et de sortie. En effet, en cas de modification de l'effet sur une des transitions concernées, il vous faudra penser à « défactoriser » et à remettre l'effet sur chaque autre transition. Un effet d'entrée (ou de sortie) doit réellement être une caractéristique de l'état dans lequel il est décrit et pas seulement un artifice local de factorisation.

ACTION D'ENVOI DE MESSAGE

N'oubliez pas de décrire dans vos diagrammes d'états l'action importante qui consiste à envoyer un message à un autre objet sur déclenchement d'une transition. La syntaxe de cette action particulière est la suivante : « /`send` cible.message ».

ÉTATS CONCURRENTS

Si un objet réalise plusieurs comportements relativement indépendants, il y a deux façons de le modéliser :

- considérer qu'il contient en fait plusieurs objets et que chacun d'entre eux réalise un de ses comportements, et représenter cela au moyen d'une classe structurée dans un diagramme de structure composite (ou un composant dans un diagramme de composants) ;
- décrire des « régions concurrentes » au sein du diagramme d'états ; l'état courant devient alors un vecteur à plusieurs lignes qui peuvent évoluer en parallèle.

COMMENT ENRICHIR LES CLASSES À PARTIR DES DIAGRAMMES D'ÉTATS ?

Des règles simples permettent d'enrichir la définition des classes à partir des diagrammes d'états :

- les opérations publiques correspondent aux noms des messages émis par les acteurs ;
- les opérations privées correspondent aux noms des messages envoyés à soi-même ;
- les attributs correspondent aux noms des données rémanentes, manipulées dans les actions, les activités ou les conditions.

OPÉRATION ET ACTEUR ?

Le concept d'opération n'a pas de sens sur un acteur humain : on ne cherche généralement pas à le modéliser d'une façon déterministe. Sur un acteur non-humain, en revanche, la liste des opérations représente son interface (au sens d'une API par exemple), telle qu'elle est utilisée par le système étudié. Cela s'avère particulièrement utile pour vérifier l'interopérabilité des deux systèmes et s'assurer que ces opérations sont déjà disponibles, ou prévues dans les spécifications.

PAS DE DIAGRAMME D'ÉTATS À MOINS DE TROIS ÉTATS !

Pas de diagramme d'états si l'on compte moins de trois états ! Ne perdez pas de temps à dessiner des diagrammes d'états qui ne contiennent que deux états (de type « on/off »),

voire un seul. Dans ce cas, la dynamique de la classe est sûrement simple et susceptible d'être appréhendée directement. En suivant cette règle, il apparaît que 10 % des classes nécessitent usuellement une description détaillée sous forme de diagramme d'états.

FAUT-IL UTILISER TOUTES LES SUBTILITÉS DU DIAGRAMME D'ÉTATS ?

Ne mettez pas forcément en œuvre toutes les subtilités des diagrammes d'états. Le formalisme du diagramme d'états UML est très puissant, mais aussi très complexe. Le lecteur qui n'en maîtrise pas tous les détails risque fort de ne pas vous suivre.