Chapitre 8

Développement du modèle dynamique

Objectifs du chapitre

Ce chapitre va nous permettre d'illustrer l'utilisation des concepts dynamiques d'UML et des diagrammes associés en phase d'analyse.

Nous verrons tout d'abord comment décrire des scénarios mettant en jeu un ensemble d'objets échangeant des messages. Ces interactions peuvent être décrites au moyen de deux types de diagrammes : le diagramme de séquence, qui met l'accent sur la chronologie des messages et le diagramme de communication (appelé collaboration en UML 1.x), qui souligne les relations structurelles des objets en interaction.

Nous détaillerons ensuite la façon de décrire le cycle de vie d'un objet d'une classe particulière, au fil de ses interactions et de son évolution propre. Le diagramme d'états permet en effet une description précise et exhaustive des états d'un objet et des transitions causées par l'arrivée d'événements, y compris les réponses de l'objet. Nous verrons donc comment utiliser efficacement les nombreux concepts des diagrammes d'états, ou *statecharts*.

Quand intervient le développement du modèle dynamique ?

Le développement du modèle dynamique constitue la troisième activité de l'étape d'analyse. Elle se situe sur la branche gauche du cycle en Y. Il s'agit d'une activité itérative, fortement couplée avec l'activité de modélisation

statique, décrite au chapitre précédent. Pour les besoins du livre, nous avons été obligés de les présenter de façon séquentielle, mais dans la réalité elles sont effectuées quasiment en parallèle. Le développement du modèle dynamique précède l'étape de conception préliminaire.

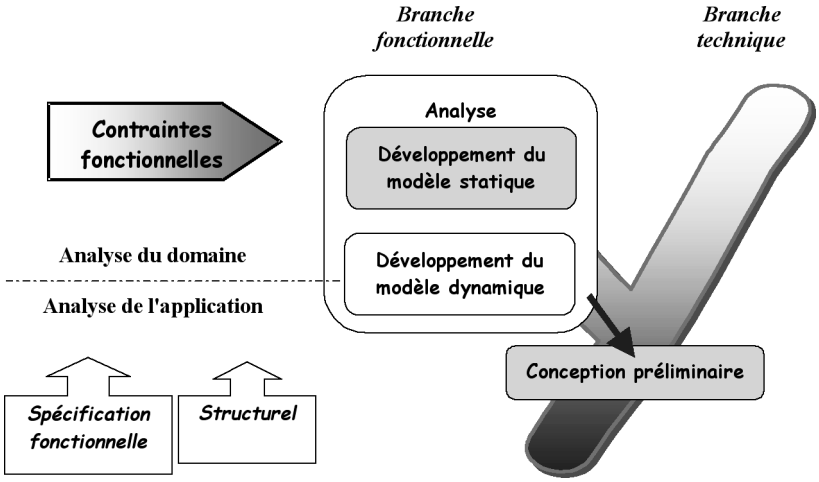


Figure 8-1 : Situation du développement du modèle dynamique dans 2TUP

Éléments mis en jeu

- Scénarios, diagrammes de séquence et de communication,
- Message, événement, signal, appel d'opération,
- Classes d'analyse de Jacobson,
- État et activité,
- Transition et condition,
- Transitions propres et internes, activités d'entrée et de sortie,
- États composites, sous-états séquentiels, sous-états parallèles.

Identifier les scénarios

Nous avons vu au chapitre 4 qu'un cas d'utilisation décrit un ensemble de scénarios. Lors de l'étape de détermination des besoins fonctionnels, un scénario représente une séquence d'interactions entre le système et ses acteurs. Le système est alors considéré comme une boîte noire. Maintenant

que nous avons développé le modèle statique d'analyse, nous allons remplacer le système par une collaboration d'objets dans chaque scénario.

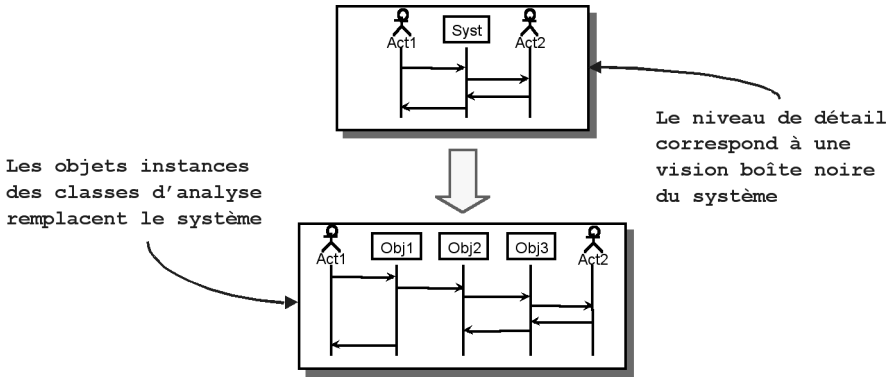


Figure 8-2 : Changement de niveau des scénarios

Un scénario décrit une exécution particulière d'un cas d'utilisation du début à la fin. Il correspond à une sélection d'enchaînements du cas d'utilisation.

On peut distinguer plusieurs types de scénarios :

- nominaux : ils réalisent les postconditions du cas d'utilisation, d'une façon naturelle et fréquente ;
- alternatifs : ils remplissent les postconditions du cas d'utilisation, mais en empruntant des voies détournées ou rares ;
- aux limites : ils réalisent les postconditions du cas d'utilisation, mais modifient le système de telle sorte que la prochaine exécution du cas d'utilisation provoquera une erreur ;
- d'erreur : ne réalisent pas les postconditions du cas d'utilisation.

Ne pas faire

NE CHERCHEZ PAS L'EXHAUSTIVITÉ DES SCÉNARIOS !

Un scénario correspond à l'exécution d'un ou de plusieurs enchaînements, joignant le début du cas d'utilisation à une fin normale ou non. Il est clair que la combinatoire des enchaînements fait exploser le nombre de scénarios potentiels ! Nous ne pourrons donc pas tous les décrire. Il faudra faire des choix, essayer de trouver le meilleur rapport « qualité/prix », c'est-à-dire l'ensemble minimal de scénarios permettant de couvrir toutes les actions/réactions du système. Cela revient à définir une famille de scénarios qui empruntent au moins une fois toutes les branches d'exécution du cas d'utilisation. Conformément à ce que nous vous avons expliqué au chapitre 4, le diagramme d'activité qui restitue ces chemins fournit un outil très efficace pour trouver les scénarios suffisants.

Ne pas faire

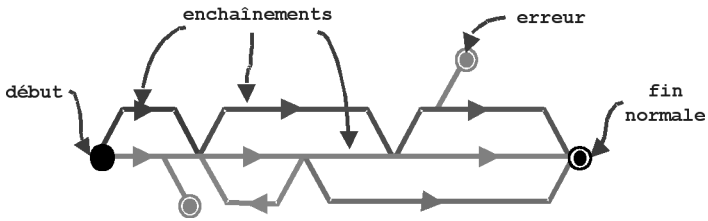


Figure 8-3 : Représentation des variantes d'un cas d'utilisation

Nous pouvons nous fixer comme objectif de couvrir toutes les exécutions importantes et réalistes du cas d'utilisation, et de faire intervenir chaque enchaînement au moins dans un scénario.

Il s'agit d'un problème analogue à celui du testeur qui doit définir des jeux de test nécessaires et suffisants, tout en sachant que l'exhaustivité n'est pas possible.

ÉTUDE DE CAS : SCÉNARIOS DE « PLANIFICATION DES MISSIONS »

Parmi tous les scénarios possibles pour le cas d'utilisation « Planification des Missions » (PM), nous avons choisi les suivants :

- scénarios nominaux :
  - PM_N1 : création d'une mission de traction validée,
  - PM_N2 : annulation d'une mission d'enlèvement en attente ;
- scénarios alternatifs :
  - PM_A1 : modification d'une mission de livraison par ajout d'une commande,
  - PM_A2 : création d'une mission d'enlèvement avec estimation incomplète ;
- scénarios aux limites :
  - PM_L1 : affectation du dernier véhicule et du dernier chauffeur de l'agence ;
- scénarios d'exception :
  - PM_E1 : non-validation de la création de mission pour cause de dépassement de tonnage,
  - PM_E2 : non-validation de la création de mission pour cause de chauffeur non qualifié,
  - PM_E3 : non-validation de la création de mission pour cause de tonnage de réserve entamé,
  - PM_E4 : essai d'annulation d'une mission moins d'une heure avant son départ.

Formaliser les scénarios

En analyse, un scénario représente un ensemble ordonné de messages échangés par des objets. On parle ici d'objet au sens large : instance de classe d'analyse ou instance d'acteur.

Le concept de message a été introduit au chapitre 3 pour le modèle de contexte dynamique. Nous allons y revenir en détail maintenant.

DIFFÉRENCE ENTRE MESSAGE, SIGNAL ET APPEL D'OPÉRATION

Un *message* représente la spécification d'une communication unidirectionnelle entre objets qui transporte de l'information avec l'intention de déclencher une réaction chez le récepteur. Il peut comprendre des paramètres qui transfèrent des valeurs de l'émetteur au récepteur. On distingue deux grandes catégories de messages :

- le *signal* : une communication asynchrone explicite et nommée entre deux objets,
- *l'appel* : l'invocation synchrone d'une opération, avec un mécanisme pour rendre ensuite la main à l'émetteur.

Au niveau logique, l'envoi d'un signal et l'appel d'une opération sont similaires. Ils impliquent tous deux une communication qui transmet de l'information par valeur d'un émetteur à un récepteur pour le faire réagir. La différence fondamentale entre les deux réside dans l'asynchronisme du signal, qui est également unidirectionnel. L'appel d'opération, au contraire, est une communication synchrone pendant laquelle le flot de contrôle passe temporairement de l'appelant à l'appelé. L'appelant perd le flot de contrôle pendant l'exécution de l'opération et le récupère à la fin de celle-ci. On peut considérer l'appel d'opération comme un signal avec un paramètre de retour implicite vers l'appelant. En analyse, nous vous conseillons d'utiliser le concept de signal, qui a une sémantique plus simple et plus générale que l'appel d'opération.[1]

1.La notation graphique des messages synchrones et asynchrones a évolué avec les versions successives d'UML, semant le trouble chez les utilisateurs et les éditeurs d'outils… Nous employons dans nos diagrammes la notation UML 2, à savoir flèche pleine pour l'appel synchrone et flèche évidée pour le message asynchrone. Comme indiqué, nous utiliserons principalement les messages asynchrones dans ce chapitre.

Les échanges de messages entre objets peuvent être représentés en UML dans deux sortes de diagrammes complémentaires :

- le diagramme de séquence, qui met l'accent sur la chronologie des messages ;
- le diagramme de communication (appelé collaboration en UML 1.x), qui souligne les relations structurelles entre les participants qui échangent les messages.

DIAGRAMME DE SÉQUENCE OU DE COMMUNICATION ?

Le diagramme de séquence et le diagramme de communication contiennent en fait le même type d'information.

Il convient alors de se poser la question suivante : quelle est la meilleure représentation visuelle pour ce que je souhaite montrer au lecteur ?

- Si je veux mettre l'accent sur l'aspect chronologique des communications, j'ai intérêt à choisir le diagramme de séquence.
- Si je veux faire ressortir les relations structurelles des participants qui interagissent, il est préférable que j'opte pour le diagramme de communication.

D'une manière générale, la plupart des auteurs considèrent qu'en analyse, le diagramme de séquence est plus apte à représenter un scénario dans le contexte d'un cas d'utilisation, et qu'en conception, le diagramme de communication se prête mieux à la représentation des itérations et des branchements complexes, ainsi que des flots de contrôle parallèles[1].

On peut également l'exprimer de la façon suivante :

- Quand il y a peu de participants mais beaucoup d'échanges entre eux, préférer le diagramme de séquence.
- Quand il y a beaucoup de participants qui interagissent, adopter le diagramme de communication.

Néanmoins, le choix est personnel et dépend aussi notablement des capacités de l'outil de modélisation utilisé. De nombreux outils du marché proposent d'ailleurs de générer automatiquement une forme à partir de l'autre, laissant complètement libre le modélisateur.

Pour illustrer les différences et la complémentarité des deux types de diagrammes d'interaction, nous allons détailler plusieurs scénarios du cas d'utilisation « Planification des missions ».

ÉTUDE DE CAS : DIAGRAMMES DE SÉQUENCE ET DE COMMUNICATION DU CAS « PLANIFICATION DES MISSIONS »

Prenons le scénario PM_N1 : « Création d'une mission de traction validée ». Nous voulons formaliser la séquence de messages suivante :

1. Cependant, les ajouts d'UML 2.0 au diagramme de séquence (en particulier les cadres d'interactions avec opérateur : loop, alt, opt, etc.) font maintenant nettement pencher la balance vers celui-ci.

- le répartiteur donne un nom d'identification et établit la nature de la mission qu'il veut créer. Comme c'est une mission de traction, il doit indiquer une agence principale de destination ;
- le répartiteur affecte les commandes à la nouvelle mission. Le système évalue au fur et à mesure des affectations le tonnage estimé de la mission ;
- le répartiteur affecte un véhicule et un chauffeur à la mission, en fonction du tonnage évalué ;
- le répartiteur valide la mission ; il doit alors préciser l'heure de départ prévue. Le système produit pour sa part une feuille de route.

Pour décrire ce scénario, nous allons faire intervenir les lignes de vie suivantes :

- un acteur *Repartiteur*,
- une mission créée au cours du scénario : *nouvelleMT:MissionTraction*,
- un élément d'une collection[1] représentant chacune des instances de *Commande* qui vont être affectées à la nouvelle mission,
- un objet *Vehicule* et un objet *Chauffeur*,
- une feuille de route créée en fin de scénario.

Le diagramme de séquence formalisant le scénario PM_N1 est présenté à la figure 8-4.

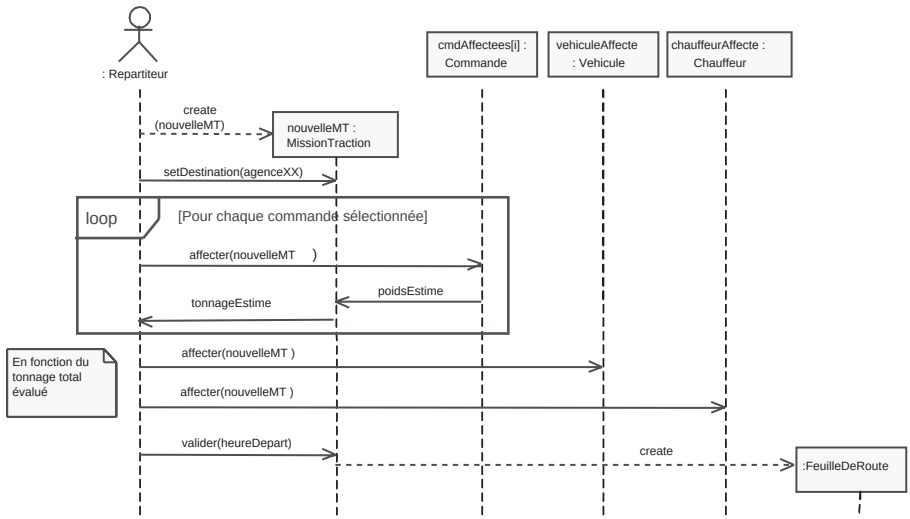


Figure 8-4 : Diagramme de séquence du scénario PM_N1

Outre les notations de base que nous supposons connues, nous avons utilisé sur ce premier diagramme des notations avancées telles que :

- la création d'une instance *nouvelleMT* (ou *FeuilleDeRoute*) au cours du scénario, figurée par le rectangle aligné en face de la flèche pointillée du message de création, et non en haut du diagramme, comme d'habitude ;
- l'itération, indiquée par le cadre avec l'opérateur loop ;
- les notes sont également très utiles pour préciser visuellement le contexte d'utilisation d'un message, ou d'un groupe de messages. Notre recommandation consiste à les regrouper

1. Notez la syntaxe avec index pour exprimer le fait que la ligne de vie représente un élément d'une collection de commandes : cmdAffectees[i].

systématiquement dans la marge gauche du diagramme.

Autre manière de formaliser le scénario : élaborer un diagramme de communication. Là encore, nous supposons les notations de base connues.

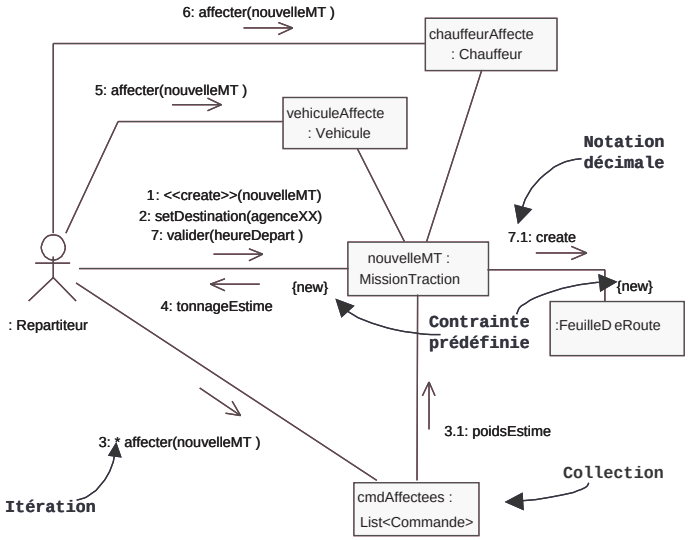


Figure 8-5 : Diagramme de communication du scénario PM_N1

Nous avons été amenés à introduire trois notations avancées :

- l'itération, indiquée par l'astérisque, permet d'indiquer graphiquement que le message `*affecter(nouvelleMT)` est envoyé à un ensemble d'objets, et non à un seul ;
- la contrainte prédéfinie {`new`} précise que l'objet est créé pendant l'exécution de l'interaction englobante et continue d'exister à la fin. Cette notation a la même signification que le rectangle en face de la flèche du message de création sur le diagramme de séquence ;
- la numérotation décimale des messages permet d'indiquer l'imbrication des appels : au lieu de numéroter en séquence (1, 2, 3...), on voit sur l'exemple que *7: valider(heureDepart)* est suivi de *7.1: create*, pour indiquer que la création est déclenchée par la validation.

Prenons maintenant le scénario PM_N2 : « Annulation d'une mission d'enlèvement en attente ». Nous voulons formaliser la séquence de messages suivante :

- le répartiteur donne un nom d'identification et établit la nature de la mission qu'il veut créer, en l'occurrence enlèvement ;
- le répartiteur affecte une première commande à la mission. Le système évalue au fur et à mesure des affectations le tonnage estimé de la mission. Il propose également l'ordre des étapes à suivre ;
- le répartiteur décide d'annuler la mission. Le système doit alors défaire les actions précédentes.

Le diagramme de séquence formalisant le scénario PM_N2 est schématisé à la figure 8-6.

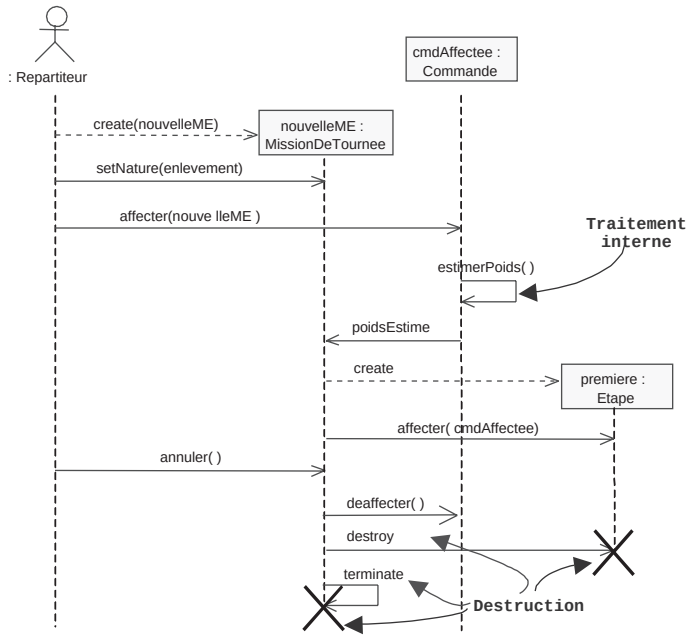


Figure 8-6 : Diagramme de séquence du scénario PM_N2

Remarquez l'utilisation de notations complémentaires sur ce diagramme de séquence :

- la flèche qui pointe vers un objet, comme *estimerPoids* sur la *Commande*, permet de représenter un traitement interne à l'objet. Même si tel n'est pas le but du diagramme d'interactions, cette indication explique plus finement le scénario, et sert surtout à préparer un éventuel diagramme d'états de la classe concernée ;
- la croix noire terminant une ligne de vie indique que l'objet est détruit pendant le scénario. Elle permet de distinguer visuellement les objets qui continuent à vivre à la fin du scénario de ceux qui ne lui survivent pas ;
- le message `destroy` est le pendant de `create`, alors que `terminate` est une convention pour indiquer l'autodestruction.

La figure 8-7 représente le diagramme de communication correspondant. Là encore, la notation décimale permet d'indiquer l'imbrication des messages. La contrainte prédéfinie {`transient`} spécifie que l'objet est créé puis détruit au cours du scénario.

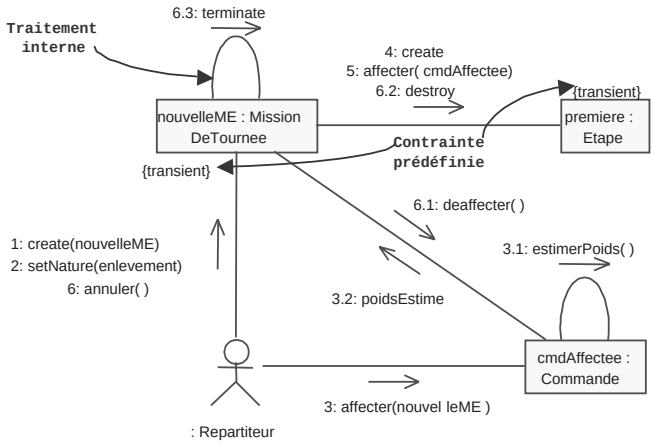


Figure 8-7 : Diagramme de communication du scénario PM_N2

Nous avons évoqué précédemment le problème incontournable de la prolifération des scénarios. Pour ne pas multiplier les diagrammes, nous vous conseillons de :

- regrouper plusieurs scénarios sur un même diagramme s'ils constituent des variantes très proches. C'est notamment le cas pour les scénarios d'exception qui se greffent généralement sur les diagrammes de séquence représentant les cas nominaux. Utilisez des notes textuelles en marge des diagrammes pour indiquer les branchements, ou les nouvelles notations UML 2 avec les cadres et les opérateurs `alt` ou `opt`. Néanmoins, si les variantes sont trop différentes, ne les regroupez pas car cela nuirait fortement à la lisibilité ;
- modéliser un enchaînement plutôt qu'un scénario complet si cet enchaînement est complexe et commun à plusieurs scénarios. Là encore, UML 2.0 a ajouté la notion de référence à une autre interaction, représentée par un cadre rectangulaire avec le mot-clé `ref`. Notez que l'idée de branchements entre scénarios est tout à fait comparable aux relations `<<include>>` et `<<extend>>` entre cas d'utilisations (voir chapitre 4). Il est d'ailleurs courant que la factorisation d'enchaînements conduise à affiner les cas d'utilisation en extrayant des cas inclus ou d'extension.

Il est clair que les techniques précédentes s'appliquent parfaitement au diagramme de séquence, mais peuvent également s'adapter au diagramme de communication. On peut par exemple utiliser un message ou une note pour faire référence au renvoi à un autre diagramme de communication.

ÉTUDE DE CAS : EXCEPTIONS DANS LE SCÉNARIO PM_N1

Nous allons compléter le diagramme de séquence du scénario PM_N1 en indiquant deux exceptions possibles : le dépassement de tonnage et le chauffeur non qualifié.

Remarquez l'utilisation de notes pour repérer l'occurrence des exceptions au cours du scénario, ainsi que la condition de poursuite de l'enchaînement nominal. Le déclenchement des exceptions est exprimé au moyen de conditions simples. Vous noterez que le diagramme ainsi complété reste parfaitement lisible, tout en contenant plus d'informations. La lisibilité est bien le critère principal pour ce type de diagramme en analyse. Nous vous conseillons d'adopter la règle pragmatique suivante : un diagramme de séquence ou de communication doit tenir sur une page A4, tout en restant lisible.

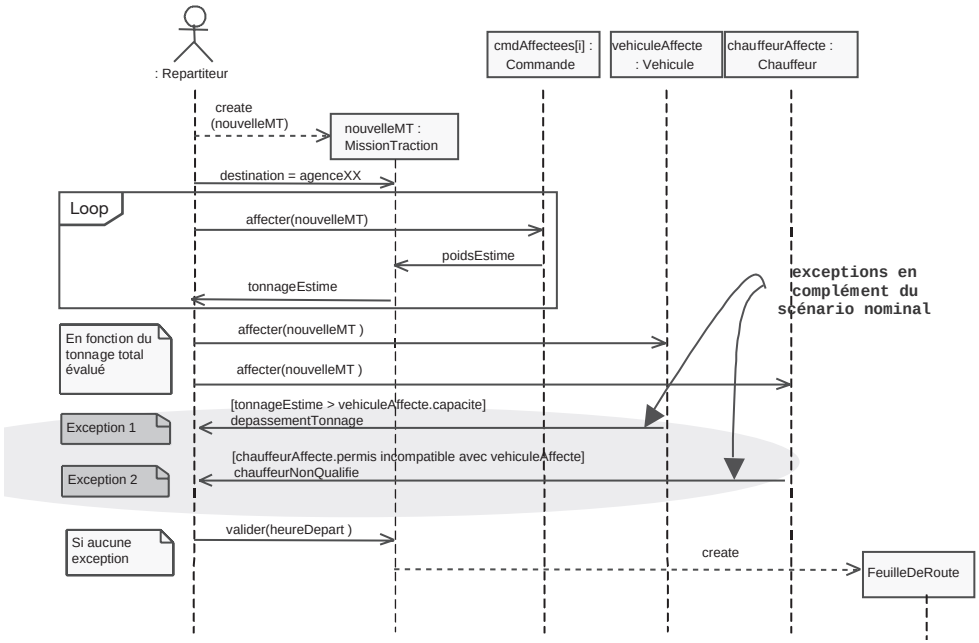


Figure 8-8 : Diagramme de séquence combinant les scénarios PM_N1, PM_E1 et PM_E2

Nous allons d'abord réaliser un diagramme de séquence « classique » du scénario nominal de création d'une nouvelle commande.

CLASSES D'ANALYSE DE JACOBSON

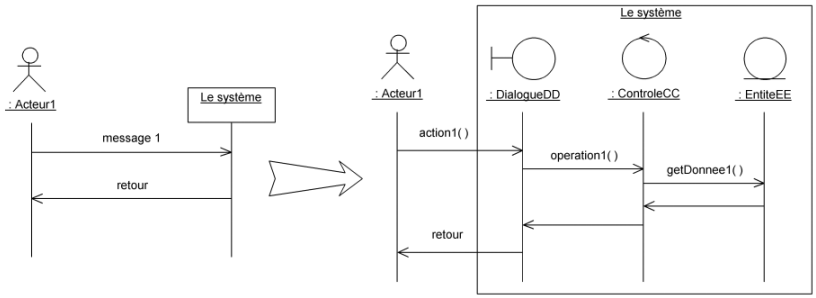
De nombreux auteurs (par exemple Conallen [Conallen 00] et Rosenberg [Rosenberg 01]), suivant les préconisations de Jacobson reprises dans le RUP, différencient dès l'analyse trois types de classes :

- Les classes qui permettent les interactions entre l'application et ses utilisateurs sont qualifiées de *boundary*. Il y a au moins une boundary pour chaque paire (acteur - cas d'utilisation).
- Celles qui contiennent la cinématique de l'application sont appelées c*ontrol*. Elles font la transition entre les boundary et les classes métier. Les control ne donneront pas forcément lieu à de vrais objets de conception, mais assurent que nous n'oublions pas de fonctionnalités ou de comportements requis par les cas d'utilisation.
- Celles qui représentent les objets métier sont qualifiées d'*entity*. Ce sont très souvent des entités persistantes, c'est-à-dire qui vont survivre à l'exécution d'un cas d'utilisation particulier.

Il existe des règles précises sur les interactions possibles entre instances de ces trois types de classes d'analyse :

- Les acteurs ne peuvent interagir (envoyer des messages) qu'avec les boundary.
- Les boundary peuvent interagir avec les control ou exceptionnellement avec d'autres boundary.
- Les control peuvent interagir avec les boundary, les entity, ou d'autres control.
- Les entity ne peuvent interagir qu'entre elles.

Le changement de niveau d'abstraction par rapport au diagramme de séquence vu au chapitre 4 peut ainsi se représenter comme sur la figure suivante, mettant en œuvre les notations graphiques proposées par Jacobson pour ses classes d'analyse.



Nous allons étudier l'utilisation de ces types de classes d'analyse sur une autre partie de l'étude de cas.

ÉTUDE DE CAS : TRAITER UNE COMMANDE

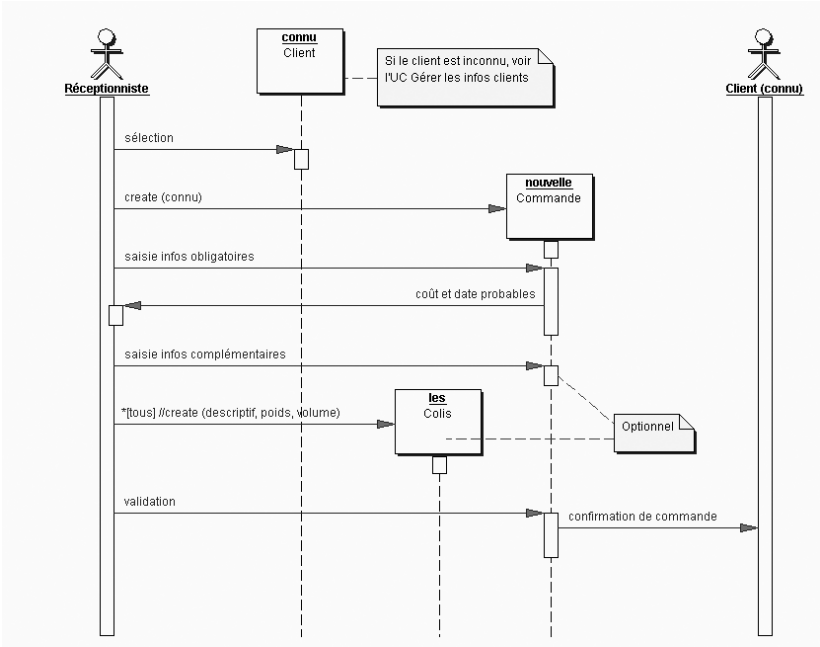


Figure 8-9 : Diagramme de séquence du scénario nominal de création d'une commande

Notez que ce diagramme (ainsi que les trois suivants) a été créé avec l'outil Together de Borland selon le standard UML 1.5. Cela explique par exemple les bandes blanches d'activation le long des lignes de vie, la notation des instances avec le souligné (supprimée par UML 2.0), ainsi que la notation de l'itération : *[tous] //create (…).

Une version complétée montrant les classes d'analyse de Jacobson (à savoir une boundary et un control) est donnée ci-après. Nous avons gardé la notation rectangulaire de l'entity par commodité et utilisé la flèche de retour en pointillés manipulée par l'outil Together.

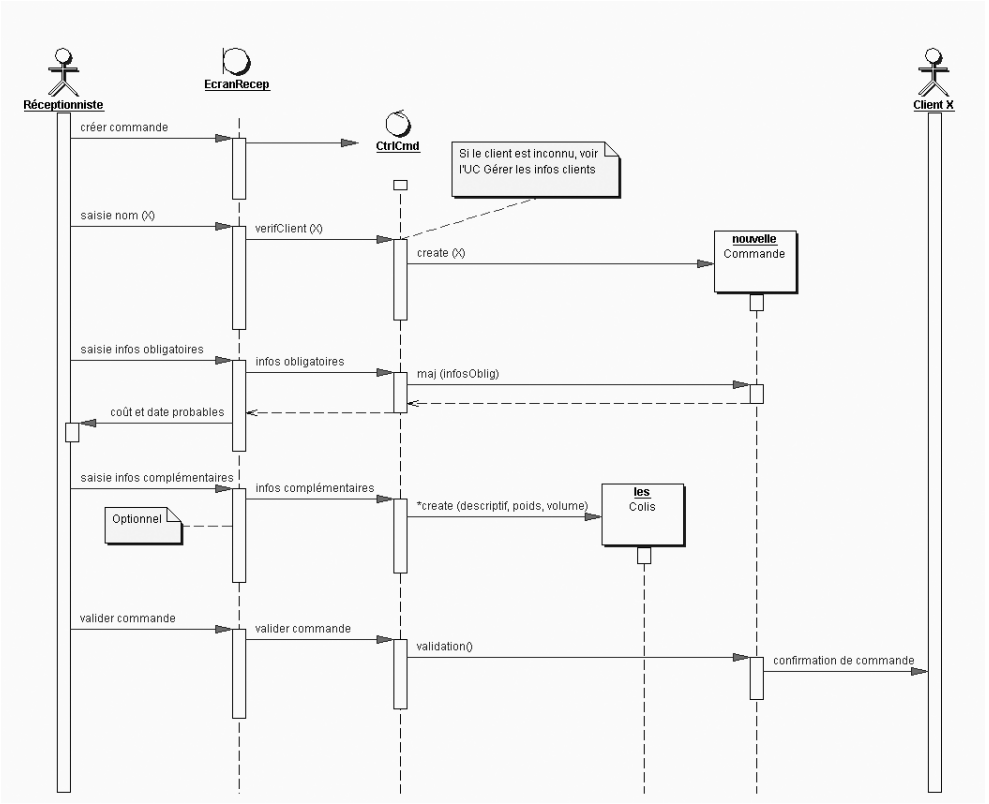


Figure 8-10 : Diagramme de séquence à la Jacobson du scénario nominal de création d'une commande

Les échanges entre la boundary et le control n'ont pas toujours une grande valeur ajoutée. En conséquence, on peut également adopter une solution intermédiaire : garder uniquement un objet control mais ne pas montrer d'objet boundary. C'est ce que nous avons réalisé sur le diagramme suivant, qui reprend le scénario nominal de création de mission. Comparez-le attentivement à celui de la figure 8-4 qui ne permettait pas bien de représenter le fait que le système fournit au répartiteur la liste des commandes à traiter et des ressources disponibles.

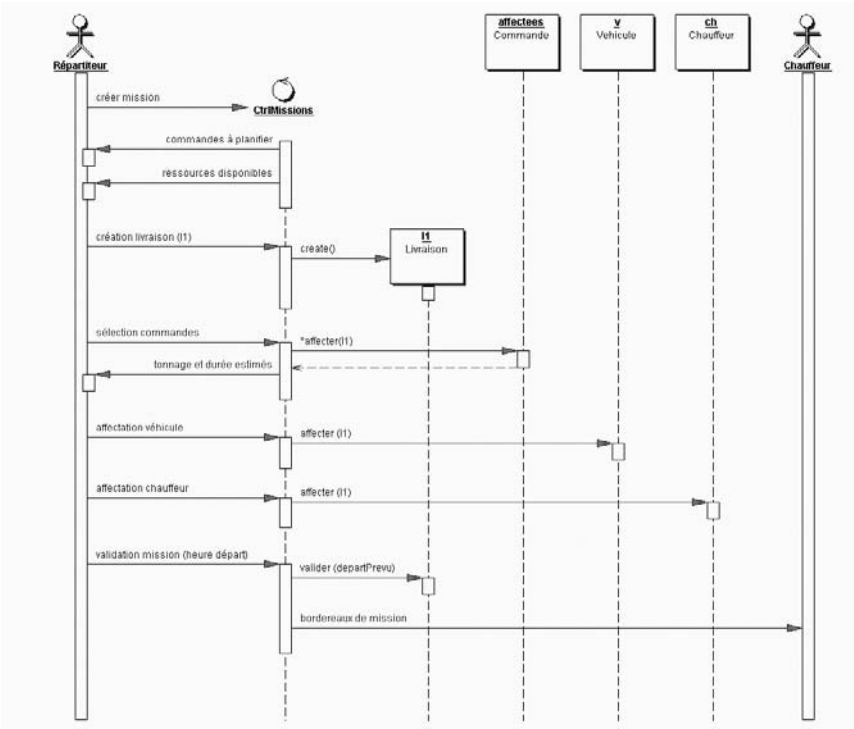


Figure 8-11 : Diagramme de séquence (avec control) du scénario nominal de création de mission

Le diagramme de communication correspondant, avec le control central caractéristique, est donné ci-après :

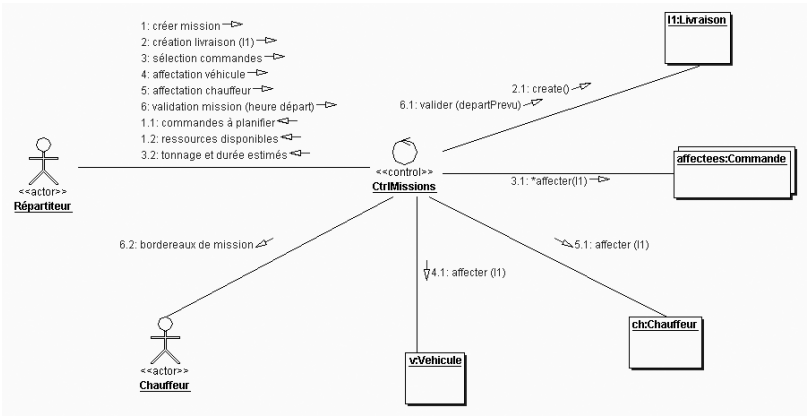


Figure 8-12 : Diagramme de communication (avec control) du scénario nominal de création de mission

Construire les diagrammes d'états

Maintenant que les scénarios ont été formalisés, la connaissance de l'ensemble des interactions entre objets permet de se représenter les règles de gestion dynamique du système. Il faut cependant se focaliser sur les classes aux comportements les plus riches de manière à développer précisément certaines de ces règles dynamiques. On recourt à cet effet au concept de *machine à états finis*, qui consiste à s'intéresser au cycle de vie d'un objet générique d'une classe particulière au fil de ses interactions avec le reste du monde, dans tous les cas possibles. Cette vue locale d'un objet, décrivant comment il réagit à des événements en fonction de son état courant et passe dans un nouvel état, est représentée graphiquement sous forme d'un *diagramme d'états*.


Définition

QU'EST-CE QU'UN ÉTAT ?

Un *état* représente une situation durant la vie d'un objet pendant laquelle :

- il satisfait une certaine condition ;
- il exécute une certaine activité ;
- ou bien il attend un certain événement.

Un objet passe par une succession d'états durant son existence. Un état a une durée finie, variable selon la vie de l'objet, en particulier en fonction des événements qui lui arrivent.

La notion d'événement mérite également d'être abordée de manière plus détaillée.


Étude

LES TYPES D'ÉVÉNEMENTS EN UML

En UML, un événement spécifie qu'il s'est passé quelque chose de significatif, localisé dans le temps et dans l'espace. Dans le contexte des machines à états finis, il représente l'occurrence d'un stimulus qui peut déclencher une transition entre états.

UML propose de distinguer plusieurs sortes d'événements :

- la *réception d'un signal* envoyé par un autre objet, ou par un acteur. Le concept d'exception, présent dans les langages C++ et Java, est un bon exemple de signal. L'envoi d'un signal est en général asynchrone ;
- l'*appel d'une opération* (*call event*) sur l'objet récepteur. L'événement d'appel est en général synchrone ;
- le *passage du temps (time event)*, qui se modélise en utilisant le mot-clé `after` suivi d'une expression représentant une durée, décomptée à partir de l'entrée dans l'état courant ;


Étude

- un *changement* dans la satisfaction d'une condition (*change event*). On utilise alors le mot-clé `when`, suivi d'une expression booléenne. L'événement de changement se produit lorsque la condition passe à vrai.

La terminaison d'une activité durable (do-activity) à l'intérieur d'un état. L'objet change alors d'état de lui-même.

Revenons maintenant aux machines à états. Une machine à états spécifie les séquences d'états qu'un objet peut parcourir durant sa vie en réponse aux événements qui lui adviennent, ainsi que les réactions correspondantes. Toutes les classes du modèle statique ne requièrent pas nécessairement une machine à états, représentée par un diagramme d'états. Il s'agit donc de trouver celles qui ont un comportement dynamique complexe nécessitant une description poussée. Cela correspond à l'un des deux cas suivants :

- les objets de la classe peuvent-ils réagir différemment à l'occurrence du même événement ? Chaque type de réaction caractérise un état particulier ;
- la classe doit-elle organiser certaines opérations dans un ordre précis ? Dans ce cas, des états séquentiels permettent de préciser la chronologie forcée des événements d'activation.


Ne pas faire

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

Ne perdez pas de temps à dessiner des diagrammes d'états contenant seulement deux états (de type « on/off »…), et encore moins un seul ! Dans ce cas, la dynamique de la classe est simple et peut être appréhendée directement.

Si vous suivez ce conseil, vous vous apercevrez que seules 10 à 20% des classes ont besoin d'une description détaillée sous forme de diagramme d'états.

Comment trouver les états d'une classe ? Pour faire un parallèle, on peut dire qu'il est aussi difficile de trouver les bons états dans le modèle dynamique que les bonnes classes dans le modèle statique !

Il n'y a donc pas de recette miracle, cependant trois démarches complémentaires peuvent être mises en œuvre :

- la recherche intuitive repose sur l'expertise métier. Certains états fondamentaux font partie du vocabulaire des experts du domaine et sont identifiables a priori (par exemple : en vol et au sol pour un avion) ;
- l'étude des attributs et des associations de la classe peut donner des indications précieuses : cherchez des valeurs seuils d'attributs qui modifient la dynamique (mineur ou majeur pour une personne), ou des comportements qui sont induits par l'existence ou l'absence de certains liens ;

- une démarche systématique peut également être utilisée : classe par classe, cherchez le diagramme d'interaction le plus représentatif du comportement des instances de cette classe, associez un état à chaque intervalle entre événements émis ou reçus par une instance et placez les transitions. Reproduisez ensuite cette démarche avec tous les scénarios faisant intervenir des instances de la classe, afin d'ajouter de nouvelles transitions ou de nouveaux états. La difficulté principale consiste à trouver ensuite les boucles dans le diagramme, afin de ne pas multiplier les états.

Comme toujours, nous vous conseillons un mélange astucieux des trois démarches.

Ensuite, pour élaborer le diagramme d'états, nous préconisons une approche incrémentale fondée sur les étapes suivantes :

- représentez d'abord la séquence d'états décrivant le comportement nominal d'un objet, de sa naissance à sa mort, avec les transitions associées ;
- ajoutez progressivement les transitions correspondant aux comportements alternatifs ;
- puis intégrez, de la même façon, celles concernant les comportements d'erreur ;
- complétez en indiquant les effets sur les transitions et les activités dans les états ;
- structurez en sous-états, si le diagramme est devenu trop complexe.

ÉTUDE DE CAS : DIAGRAMME D'ÉTATS DE LA CLASSE *MISSION*

Nous allons illustrer cette approche incrémentale de construction du diagramme d'états en prenant l'exemple concret de la classe *Mission*. Pour cela, nous nous baserons sur les diagrammes d'interactions déjà réalisés, ainsi que sur l'étude des scénarios des cas « Planification des Missions » et « Suivi des Missions ».

Si l'on considère les diagrammes d'interactions du scénario nominal PM_N1, on peut déjà identifier deux états importants :

- *En attente* : de la création d'une instance de mission à sa validation, cet état inclut toutes les opérations d'affectation de commandes et de ressources ;
- *Validee* : dans le cas nominal, les affectations se sont bien passées et le répartiteur valide la mission, qui attend alors son départ effectif.

Pour terminer la séquence d'états représentant le comportement nominal d'une *Mission*, nous allons ajouter un état *En cours*, et un état *Terminee*. Nous obtenons alors un premier squelette de diagramme d'états, présenté sur la figure 8-13.

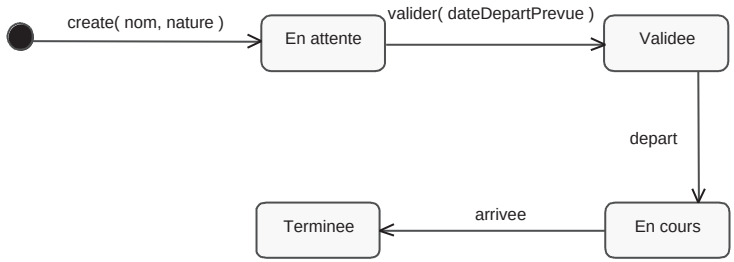


Figure 8-13 : Transitions nominales du diagramme d'états de la classe Mission

Ajoutons maintenant les transitions correspondant aux comportements alternatifs :

- la possibilité de modifier une mission en attente, ou déjà validée ;
- la possibilité d'annuler une mission validée, mais pas encore partie. Nous sommes ainsi amenés à introduire un état final.

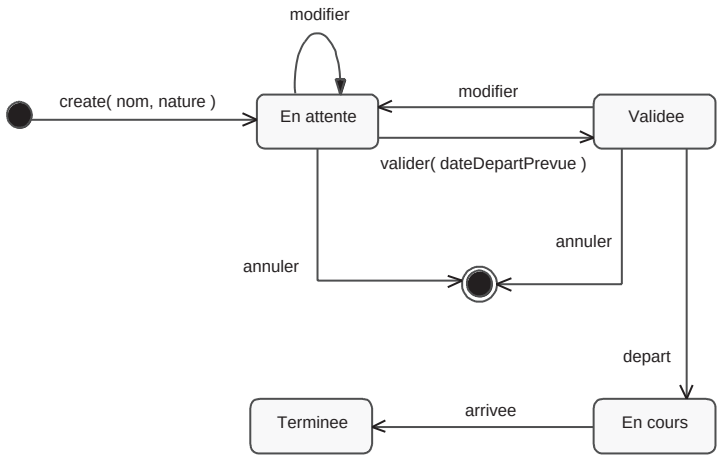


Figure 8-14 : Ajout des transitions alternatives sur le diagramme d'états de la classe Mission

Notez également l'utilisation de la transition propre, qui pointe vers l'état *En attente*.

En fait, on ne peut plus ni modifier ni annuler une mission moins d'une heure avant le départ prévu. Une première solution consiste à ajouter une condition de garde sur les transitions correspondantes, comme illustré à la figure 8-15.

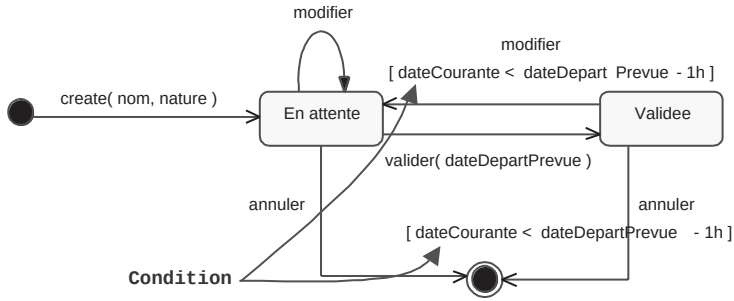


Figure 8-15 : Exemples de conditions sur le diagramme d'états de la classe Mission

Pour éviter de dupliquer la condition [*dateCourante < dateDepartPrevue -1h*], il est également possible de la remplacer par un état intermédiaire entre *Validee* et *En cours*, que nous appellerons *Non modifiable*. Mais quel événement déclencherait la transition entre *Validee* et *Non modifiable* ? Il ne s'agit pas d'un événement déclenché par la réception d'un message, mais d'un changement interne à l'objet lui-même. La notion de change *event* évoquée précédemment répond à ce besoin. La transition est donc déclenchée par l'occurrence de *when* (*dateCourante < dateDepartPrevue -1h*).

Si nous voulons exprimer le fait qu'au bout de 48 h une mission terminée est archivée et supprimée de la mémoire vive du système, quelle est la marche à suivre ? La notion de *time event* évoquée précédemment répond à ce besoin. La transition de passage dans l'état final est donc déclenchée par l'occurrence de `after` (48 h).

Le diagramme d'états de la classe *Mission* devient alors celui de la figure 8-16.

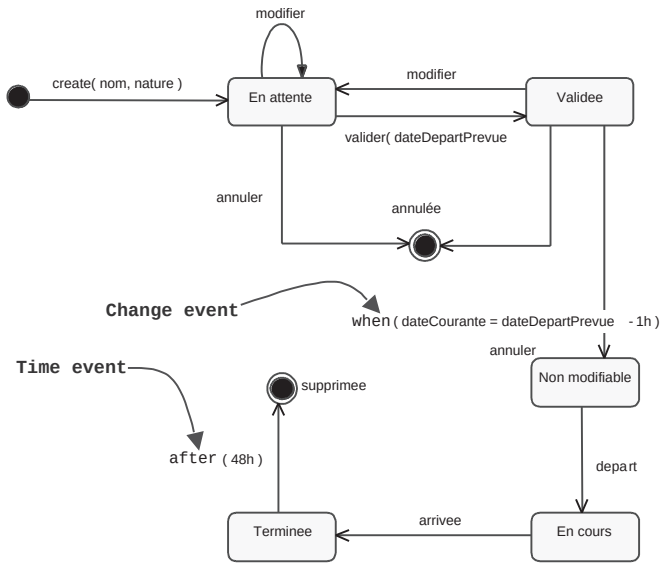


Figure 8-16 : Nouveaux ajouts sur le diagramme d'états de la classe Mission

Il est possible d'indiquer plusieurs états finaux avec des noms différents, afin d'augmenter la lisibilité du diagramme, ou simplement de distinguer des manières différentes de détruire un objet.

Pour l'instant, le diagramme d'états que nous avons dessiné représente bien la chronologie possible des événements qui peuvent affecter les objets de la classe *Mission*, avec les différents états correspondants. En revanche, il lui manque une partie importante : comment réagissent à leur tour ces objets lors des changements d'états ?

UML a prévu deux possibilités pour décrire les réactions d'un objet, et plus généralement ses traitements et ses opérations :

- les *effets* associés aux transitions sont considérés comme atomiques, c'est-à-dire ininterruptibles, ou encore instantanés par rapport à l'échelle de temps considérée. Ils peuvent représenter des appels d'opérations sur d'autres objets, ou sur l'objet lui-même, la création ou la destruction d'un autre objet, ainsi que l'envoi d'un signal à un autre objet ;
- les *activités durables (do-activity)*, au contraire, ont une certaine durée, sont interruptibles et sont donc associées aux états. On distingue les activités continues (qui ne s'arrêtent qu'à l'arrivée d'un événement faisant sortir l'objet de l'état courant) et les activités finies (qui s'arrêtent de toute façon par elles-mêmes au bout d'un certain temps, ou lorsqu'une certaine condition est remplie).

Illustrons-le en ajoutant des effets et des activités sur le diagramme d'états de la classe *Mission*.

ÉTUDE DE CAS : AJOUT D'EFFETS ET D'ACTIVITÉS POUR LA CLASSE MISSION

La définition d'une mission, à savoir l'affectation des commandes, d'un véhicule et d'un chauffeur, constitue une activité durable. En revanche, la création d'une FeuilleDeRoute ou d'un SuiviMission, l'envoi d'un signal au répartiteur ou l'archivage de la Mission peuvent être considérés comme de simples effets. Remarquez la notation particulière proposée par UML pour l'envoi de signal, avec le mot-clé `send`.

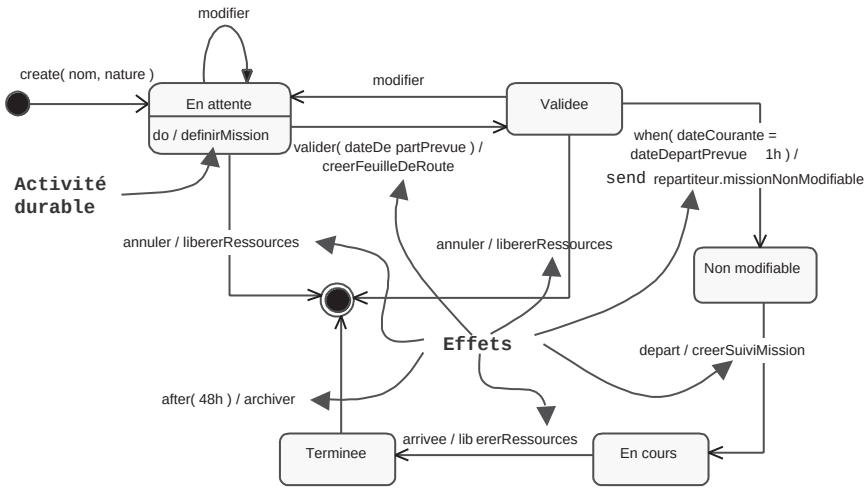


Figure 8-17 : Ajout d'effets et d'activités sur le diagramme d'états de la classe Mission

Si nous voulons factoriser l'effet *libererRessources* qui se retrouve sur trois transitions, UML nous propose un raccourci sous la forme d'un effet d'entrée déclenché par le pseudo-événement `entry`. À la figure 8-18, nous avons simplifié le diagramme d'états de la classe Mission en supprimant provisoirement l'état *Terminee* ainsi que la transition déclenchée par l'événement temporel *after(48h)*. Par conséquent, les trois transitions qui conduisent à l'état final doivent porter l'effet *libererRessources*. Celui-ci peut alors être factorisé en entrée de l'état final :

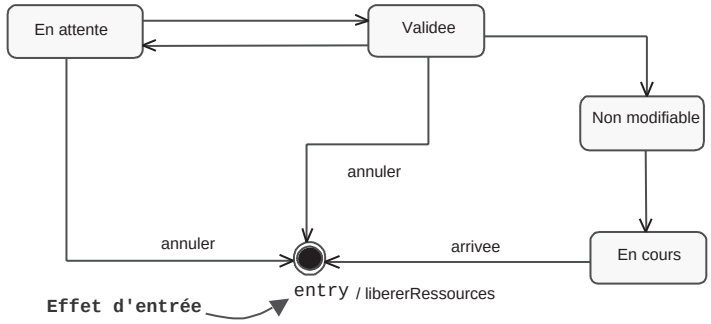


Figure 8-18 : Exemple d'effet d'entrée sur un état final

La même notation existe bien sûr également en sortie d'un état : `exit`.

Revenons maintenant sur la notion importante d'activité durable. Quand un objet se trouve dans un état, soit il reste passif, et attend qu'un événement le fasse changer d'état, soit il exécute une activité durable jusqu'à ce qu'elle soit interrompue par un événement. Par exemple, dans l'état *En attente*, on peut considérer que l'objet *Mission* se définit progressivement, jusqu'à ce que le répartiteur décide de le valider. L'activité *definirMission* recouvre un traitement complexe consistant à enchaîner les affectations de commandes, celles de ressources et les créations d'étapes. Elle nous permet de rester à

un niveau d'abstraction élevé avant d'entrer dans le détail. Si nécessaire, on peut ensuite préciser l'activité en référençant un autre diagramme montrant de façon détaillée l'intérieur du traitement encapsulé par l'activité.

La figure 8-19 illustre cette manière de décrire une activité de haut niveau. Vous remarquerez que les transitions entre états ne sont pas forcément déclenchées par des événements. Une transition déclenchée implicitement lorsqu'un état a terminé son activité finie est appelée *transition automatique.*

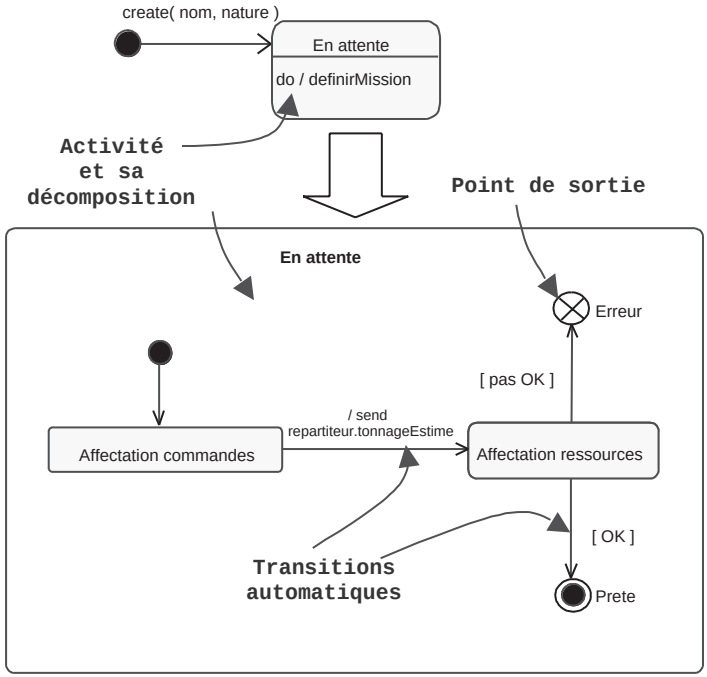


Figure 8-19 : Sous-diagramme de l'opération definirMission

Le déroulement nominal de definirMission consiste en l'enchaînement des deux activités affectation commandes et affectation ressources, avec un résultat OK pour cette dernière. Notez qu'une transition automatique peut porter une condition de garde, ainsi qu'une activité. Notez également l'utilisation du nouveau concept UML 2.0 de point de sortie (*exit point*) permettant de distinguer une sortie spécifique (condition [pas OK]) de celle correspondant à la fin normale de l'activité.

Comment savoir au niveau du diagramme d'états supérieur que l'activité *definirMission* est en fait décrite par un sous-automate ? Là encore, UML propose une notation particulière, appelée indicateur de décomposition cachée, comme indiqué à la figure 8-20 :

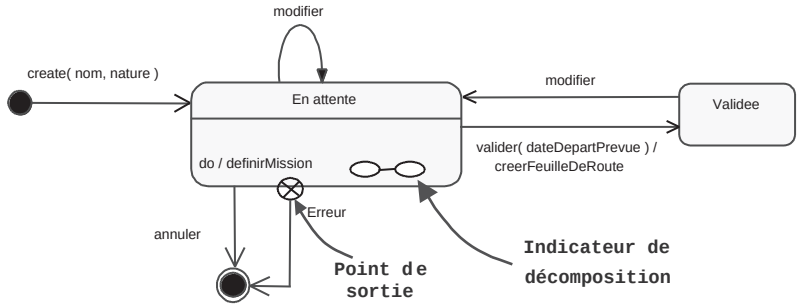


Figure 8-20 : Exemple d'indicateur de décomposition

Remarquez que l'on retrouve la notation du point de sortie permettant d'exprimer le fait qu'à partir du sous-état erreur, l'objet passe directement dans l'état final

Mais que se passe-t-il lors de l'occurrence de l'événement *modifier* ? Puisqu'il s'agit d'une transition propre, l'objet retourne dans l'état *En attente*, mais comme celui-ci est détaillé par un autre diagramme, il se retrouve en fait dans le sous-état initial, à savoir *Affectation commandes*. Ce qui est bien sûr inutile : une modification d'affectation de ressources obligerait à repasser en affectation de commandes. Pour traiter un événement qui ne modifie pas l'état courant de l'objet, UML propose un concept particulier : la transition interne, notée à l'intérieur de l'état, comme à la figure 8-21.

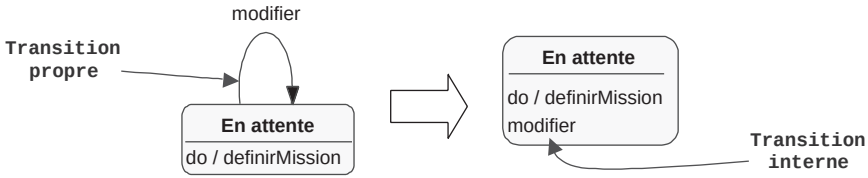


Figure 8-21 : Transition propre et transition interne

Une transition interne contient généralement un effet, comme sur l'exemple suivant, qui nous permet de détailler les traitements à entreprendre en cas d'affectation ou désaffectation d'une commande.

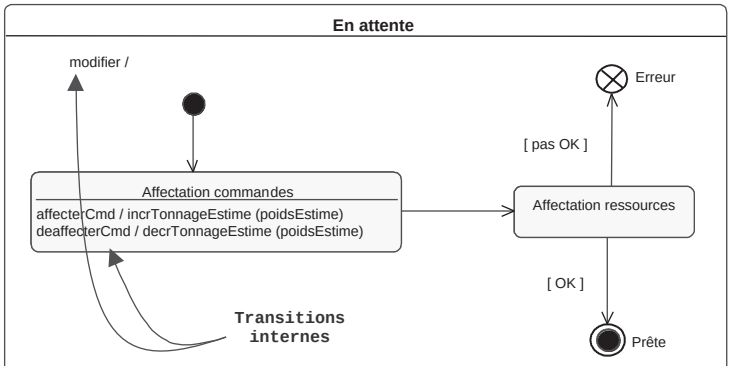


Figure 8-22 : Transitions internes avec effet


Conseil

PRÉFÉREZ LES TRANSITIONS INTERNES AUX TRANSITIONS PROPRES !

Les transitions internes présentent une différence subtile avec les transitions propres. En effet, lors d'une transition propre, l'objet quitte son état courant, déclenchant un éventuel effet de sortie (`exit`), puis retourne dans ce même état, et engendre le cas échéant un effet d'entrée (`entry`). De plus, l'activité durable à l'intérieur de l'état est interrompue, puis redémarrée. Au contraire, la transition interne n'a aucun effet de bord.


Conseil

Nous avons également évoqué le fait que la transition propre ramène systématiquement dans le sous-état initial si l'état auquel elle s'applique est affiné par un autre diagramme.

Le diagramme d'états consolidé de la classe *Mission* est maintenant suffisamment complexe pour qu'il devienne indispensable d'introduire la notion d'état composite.


Définition

QU'EST-CE QU'UN ÉTAT COMPOSITE ?

Un *état composite* est un état qui contient d'autres états, appelés sous-états ou états imbriqués. Ces derniers peuvent être :

- *séquentiels* (ou encore disjoints),
- ou *concurrents* (aussi appelés parallèles).

Les sous-états sont à leur tour susceptibles d'être décomposés.

ÉTUDE DE CAS : AJOUT DE SOUS-ÉTATS POUR LA CLASSE *MISSION*

Nous avons déjà vu des sous-états séquentiels avec la décomposition de l'activité *definirMission*. La notation particulière de l'indicateur de décomposition cachée (figure 8-20) permet de rester au niveau d'abstraction supérieur. Il est cependant tout à fait possible d'inclure le diagramme montrant les sous-états à l'intérieur de l'état englobant.

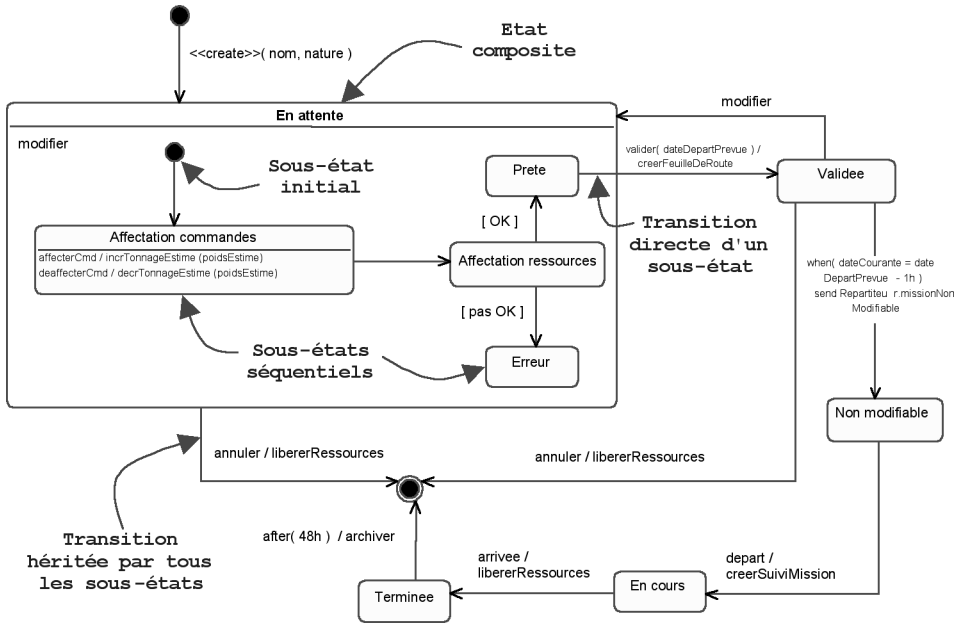


Figure 8-23 : Sous-états séquentiels

Le sous-état initial (*Affectation commandes*) indique quel sous-état séquentiel est atteint lors de l'entrée dans l'état englobant *En attente*. C'est le cas à la création, mais aussi lors d'une modification alors que la mission a déjà été validée. Notez ensuite la différence entre les deux transitions suivantes :

- celle qui est déclenchée par *annuler*, et qui part du contour de l'état englobant *En attente* : elle est héritée par chacun des sous-états ;
- celle induite par *valider*, et qui sort directement du sous-état *Prete*, en traversant le contour de l'état englobant : elle est spécifique au sous-état et non pas héritée.

Si nous voulions maintenant détailler l'affectation des ressources, nous pourrions décrire plus avant le processus mis en œuvre par le répartiteur :

- chercher un véhicule libre de tonnage suffisant,
- trouver un chauffeur qualifié pour ce véhicule.

Pour gagner du temps, il se peut que le répartiteur veuille effectuer ces deux recherches en parallèle, surtout s'il sait que la plupart de ses chauffeurs sont qualifiés pour tous les véhicules. UML permet de décrire deux activités parallèles à l'intérieur de l'état *Affectation ressources*, comme illustré à la figure 8-24.

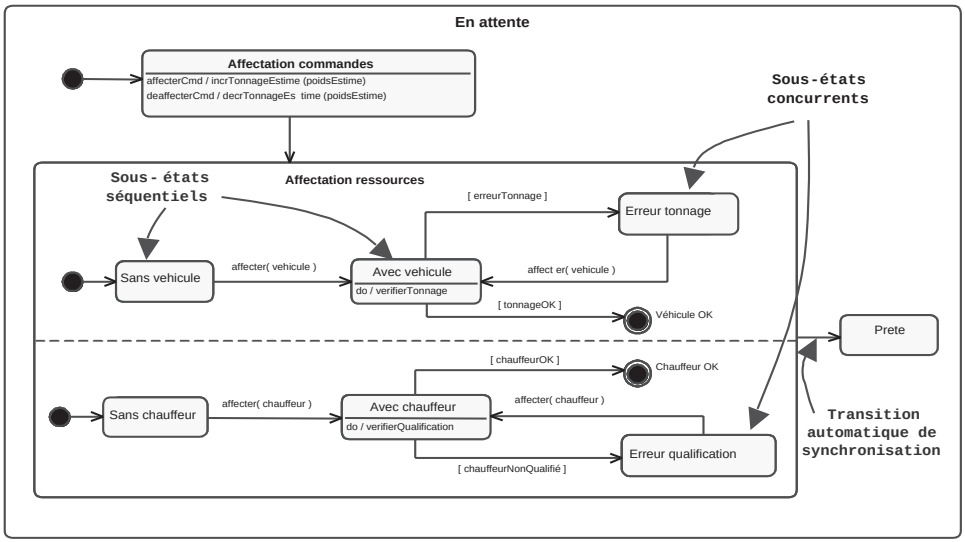


Figure 8-24 : Exemple de sous-états concurrents

Les traits pointillés indiquent la séparation en deux sous-automates concurrents.

Notez qu'il faut initialiser chacun des sous-automates : l'état initial est donc composé du couple (*Sans vehicule*, *Sans chauffeur*). Normalement, l'état final à l'intérieur d'un état composite permet de spécifier que l'exécution de l'activité de haut niveau est terminée. Quand il est atteint, une transition automatique partant de l'état englobant est automatiquement déclenchée. Dans le cas des sous-automates parallèles, c'est un peu plus compliqué : il faut que les deux activités soient terminées, dans n'importe quel ordre. Dans notre exemple, la transition qui part de l'état englobant *Affectation ressources* vers l'état *Prete* est dite de synchronisation, car elle ne sera déclenchée que lorsque les deux sous-automates seront chacun arrivés dans son état final.

Ne pas faire

NE VOUS CROYEZ PAS OBLIGÉ D'UTILISER TOUTES LES SUBTILITÉS DES DIAGRAMMES D'ÉTATS !

Comme vous avez pu le constater sur le petit exemple de la classe *Mission*, le formalisme du diagramme d'états UML est très puissant, mais aussi très complexe ! Et encore, nous n'avons pas évoqué :

- le pseudo-état *history*, dont l'activation renvoie au dernier sous-état actif d'un état composite ;
- les événements différés (`defer`) qui sont mémorisés pour être traités dans un état futur qui les acceptera, etc.

Gardez à l'esprit que la plupart des diagrammes d'états peuvent être exprimés de façon satisfaisante avec les concepts de base : état, transition, condition, effet, activité.

Toutes les astuces de notation sont séduisantes, mais nécessitent en contrepartie une expertise certaine de la part du lecteur, d'autant que les concepts avancés sont ceux qui ont subi la plus grande évolution depuis les débuts d'UML, et qui sont les moins bien pris en charge par les outils...

Valider les diagrammes d'états avec les diagrammes d'interactions

Nous avons insisté précédemment sur la complémentarité entre :

- les diagrammes d'interaction (séquence et communication),
- et les diagrammes d'états.

Cette complémentarité est fondamentale ; il est important de confronter ces deux points de vue, dès que l'on dispose des diagrammes correspondants.

En effet, les diagrammes d'états apportent précision et exhaustivité, et permettent de valider et de compléter les diagrammes d'interactions. Ils peuvent également inciter à créer de nouveaux diagrammes d'interactions pour compléter ceux qui existent déjà. Il faut toutefois vérifier que les diagrammes d'états des classes impliquées dans les diagrammes d'interactions prennent bien en compte tous les scénarios décrits, et qui plus est de façon correcte.

Le schéma suivant représente les relations entre les deux types de diagrammes. Prenons un diagramme de séquence simple mettant en jeu deux objets : *a* de la classe *A* et *b* de la classe *B*. L'interaction entre ces deux objets consiste en l'enchaînement de deux événements :

- suite à la réception de l'événement *ev0*, *a* envoie *ev1* à *b* ;
- en retour, *b* envoie *ev2* à *a*.

Les diagrammes d'états des classes A et B doivent forcément être cohérents avec cette interaction, même s'ils intègrent de nombreux autres comportements.

Nous les avons représentés partiellement le long de la ligne de vie de chaque objet. Ainsi, l'objet *a* est dans l'état *EA0* avant de recevoir *ev0*, puis passe dans le nouvel état *EA1*, après avoir envoyé *ev1* à *b*. L'événement *ev1* doit être admissible à ce moment-là par l'automate de la classe B, qui va à son tour changer d'état après avoir répondu *ev2* à *a*. De nouveau, cela signifie que, dans l'état *EA1*, *a* doit être capable de traiter *ev2*. Nous voyons donc que l'on doit être capable de suivre toutes les interactions entre objets sur les diagrammes d'états des classes concernées ; ce sont des contraintes à vérifier.

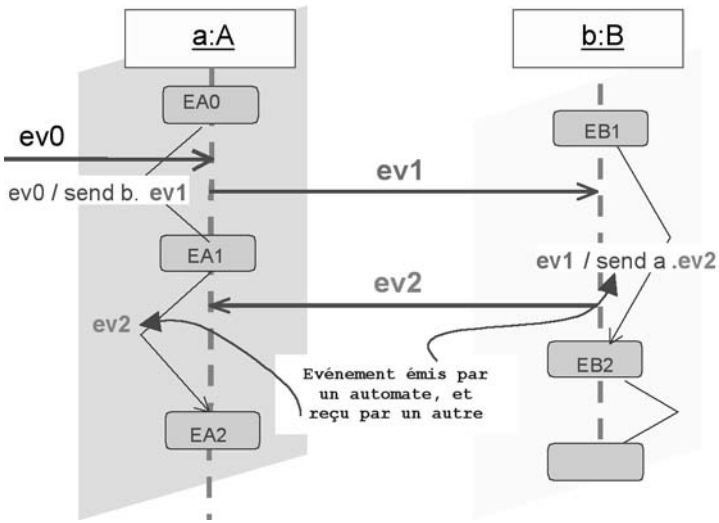


Figure 8-25 : Complémentarité entre diagrammes de séquence et d'états

Confronter les modèles statique et dynamique

Nous avons évoqué au fil des chapitres 4, 7 et 8, les relations diverses qui existent entre les principaux concepts du modèle statique (objet, classe, association, attribut et opération) et les principaux concepts dynamiques (message, événement, état et activité).

Les correspondances sont loin d'être triviales, car il s'agit bien de points de vue complémentaires, et non redondants. Essayons de synthétiser les plus importantes, sans prétendre à l'exhaustivité :

- un message peut être un appel d'opération sur un objet (le récepteur) par un autre objet (l'émetteur) ;
- un événement ou un effet sur une transition peuvent correspondre à l'appel d'une opération ;
- une activité dans un état peut concerner l'exécution d'une opération complexe ou d'une succession d'opérations ;
- un diagramme d'interactions met en jeu des objets (ou des rôles) ;
- une opération peut être décrite par un diagramme d'interaction ou d'activité ;
- une condition de garde et un *change event* peuvent consulter des attributs ou des liens statiques ;

- un effet sur une transition peut manipuler des attributs ou des liens statiques ;
- le paramètre d'un message peut être un attribut ou un objet entier.

ÉTUDE DE CAS : ILLUSTRATION DES CORRESPONDANCES ENTRE MODÈLES STATIQUE ET DYNAMIQUE SUR LA CLASSE *MISSION*

Commençons par les paramètres des messages. L'exemple suivant montre que :

- le paramètre *dateDepartPrevue* (que l'on retrouve aussi ailleurs dans une condition de garde) correspond à un attribut de la classe ;
- le paramètre nom du message de création correspond probablement à l'attribut *reference*. Il faut donc mettre à jour l'un des deux diagrammes pour harmoniser ;
- le paramètre nature du message de création correspond en fait à la combinaison de la spécialisation et de l'attribut *nature* de la sous-classe *MissionDeTournee*.

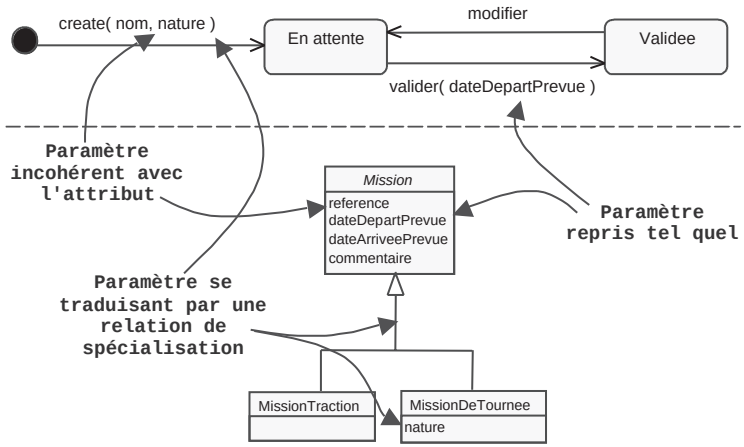


Figure 8-26 : Exemples de correspondances entre paramètres de messages et attributs

Un deuxième exemple va nous permettre d'observer d'autres correspondances :

- l'affectation *setDestination(agenceXX)* illustre le fait qu'une activité sur une transition peut utiliser des attributs ou des liens. Ici, l'affectation ne met pas simplement à jour un attribut de l'objet, elle crée un lien avec un objet de la classe *Agence* qui prend le rôle de destination par rapport à l'objet concerné de la classe *Mission* ;
- le message *tonnageEstime* correspond au résultat d'un traitement réalisé par l'objet *Mission*. Il s'agit donc d'une responsabilité de la classe *Mission*, qui devrait se traduire ici à la fois par une opération de calcul et un attribut pour mémoriser le résultat ;
- le message *valider(dateDepartPrevue)* correspond directement à l'invocation de l'opération valider sur l'objet *Mission* concerné, avec comme paramètre un des attributs déjà répertoriés.

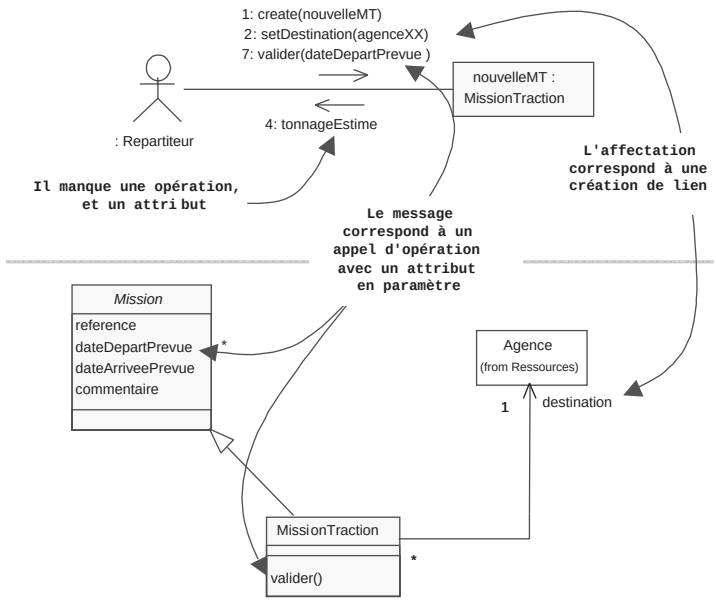


Figure 8-27 : Deuxième exemple de correspondances statique/dynamique

Conseil

COMPLÉTEZ LES DIAGRAMMES DE CLASSES AVEC LES ATTRIBUTS ET OPÉRATIONS IDENTIFIÉES GRÂCE À L'ANALYSE DYNAMIQUE !

Il ne faut pas oublier de mettre à jour les diagrammes de classes, afin de profiter de l'analyse réalisée avec les différents diagrammes dynamiques.

Vérifiez également la bonne affectation des opérations : peut-être faut-il les remonter s'il existe une super-classe.

ÉTUDE DE CAS : COMPLÉMENTS SUR LA CLASSE *MISSION*

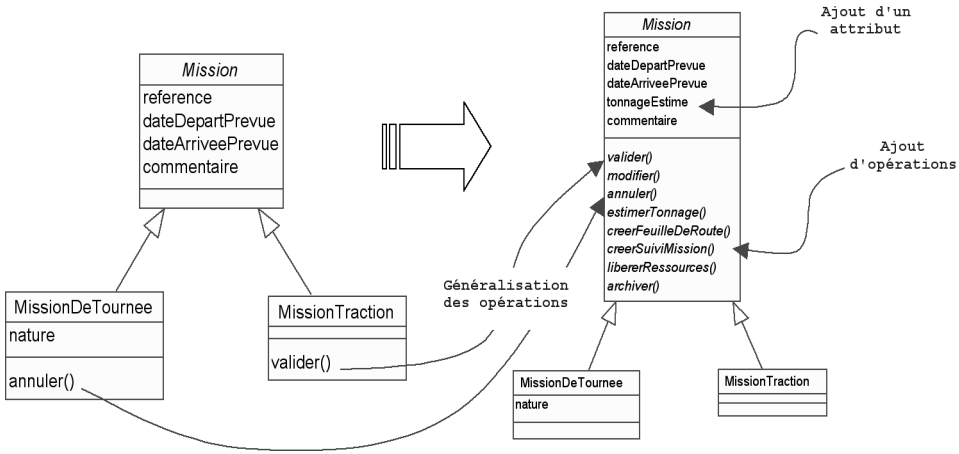


Figure 8-28 : Compléments sur la classe Mission

Conseil

FONDEZ-VOUS SUR LE CONCEPT DE STABILITÉ POUR CHOISIR ENTRE STATIQUE ET DYNAMIQUE !

La notion de stabilité est très importante en modélisation : on représente par exemple les éléments les plus stables par des classes, et d'autres moins stables par des états ou des messages. On peut également dire que les associations décrivent des liens stables entre classes et les messages des relations dynamiques entre objets.

Phases de réalisation du modèle dynamique d'analyse

Deux techniques permettent de modéliser la dynamique d'un système avec UML.

La première consiste à décrire comment des instances des classes d'analyse communiquent entre elles pour produire un comportement global du système. On parle dans ce cas de collaboration entre objets, comprenant à la fois :

- une vue structurelle constituée des objets et des liens statiques entre ces objets,
- une vue dynamique appelée interaction, composée par les flots de messages entre objets circulant sur les liens statiques.

Avec la deuxième, on s'intéresse au cycle de vie d'un objet d'une classe particulière au fil de ses interactions avec le reste du monde, dans tous les cas possibles. Cette vue locale d'un objet s'appelle une machine à états, décrivant comment l'objet réagit à des événements en fonction de son état courant et passe dans un nouvel état.

L'ensemble des cas d'utilisation découverts lors de la capture des besoins fonctionnels guide toutes les vues dynamiques, en structurant les scénarios décrits par des interactions.

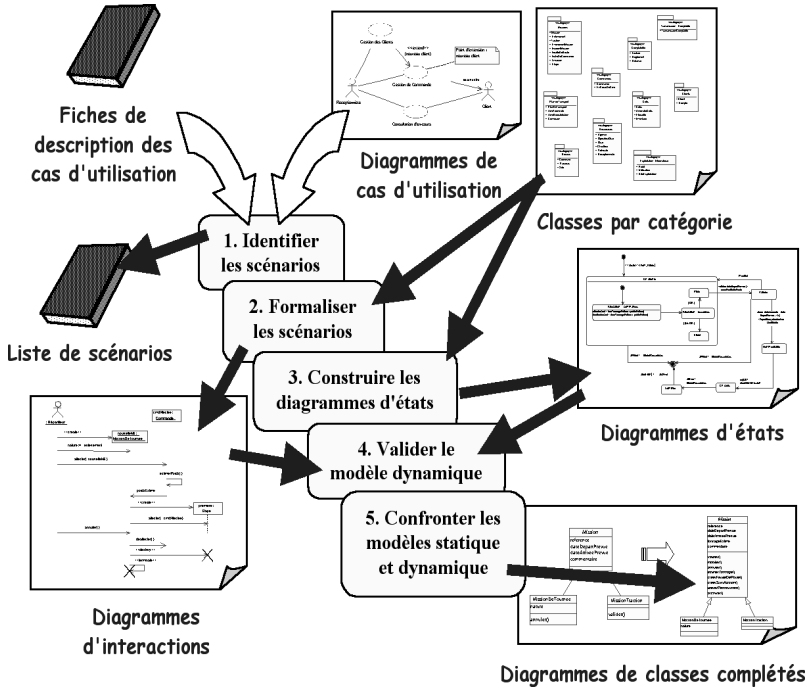


Figure 8-29 : Démarche d'élaboration du modèle dynamique

Comme nous l'avons dit au début de ce chapitre, les modèles statique et dynamique sont très fortement couplés et doivent se construire par itérations successives, chacun complétant l'autre.

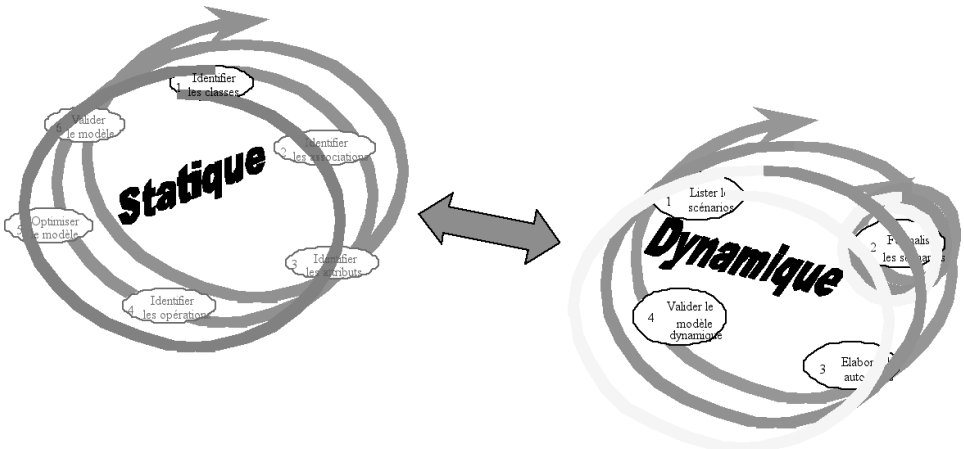


Figure 8-30 : Couplage entre les démarches des modèles statique et dynamique