**Ressource : éléments sur quelques diagrammes SysML**

# Eléments sur les diagrammes de cas d’utilisation

• Les **cas d’utilisation (use cases)** expriment les services rendus par le système (et pas la façon dont ils sont réalisés) à un niveau élevé de description (pas de détails). Ils s’énoncent généralement par des verbes à l’infinitif. Les cas d’utilisation sont regroupés dans un ou plusieurs diagrammes.

|  |  |
| --- | --- |
| **Acteurs :** | |
|  | Un **acteur** est qualifié de **principal** pour un **cas d'utilisation** (inscrit dans un ovale) lorsque ce cas lui rend un service.  • Le système, délimité par sa **frontière**, est développé pour rendre des services aux acteurs principaux.  • Les autres acteurs sont qualifiés de **secondaires**. Ils contribuent à la réalisation des services.  • Chaque cas d’utilisation est relié par une ligne aux acteurs impliqués. |
|  | **Généralisation / spécialisation entre acteurs :**  • L’acteur B est une généralisation de l’acteur A.  • Autrement dit, A est une spécialisation de B.  • A hérite des cas d’utilisation de B (tous les cas d'utilisation accessibles à B le sont aussi à A, mais l'inverse n'est pas vrai). |
| **Relations standards entre cas d’utilisation :** | |
|  | **Inclusion :**  • Le cas A inclut le cas B.  • Autrement dit, B est une partie de A.  • B est toujours exécuté au cours de l’exécution de A. |
|  | **Extension :**  • Le cas A étend le cas B.  • A est exécuté lors de l’exécution de B si une condition est remplie.  • L’exécution de A est donc optionnelle. |
|  | **Généralisation / spécialisation entre cas d’utilisation :**  • Le cas B est une généralisation du cas A.  • Autrement dit, A est une spécialisation (cas particulier) de B.  • A hérite des acteurs de B. |

# Eléments sur les diagrammes états-transitions

*La forme spécifique des machines à états finis utilisée dans UML-SysML est basée sur une variante orientée objet du formalisme des diagrammes états-transitions (statecharts) de David Harel. Cependant, il existe un petit nombre de différences sémantiques qui distinguent la version UML-SysML de l'original. Les indications fournies ici sont basées sur la version 1.6 de SysML, la version 2.5.1 d’UML, et la version 1.0 du Precise Semantics of UML State Machines (PSSM). Seules sont présentées les machines à états dites de comportement.*

**A) Concepts de base**

**Machine à états (state machine) – Diagramme états-transitions (statechart)**

• Une **machine à états** ou **machine d’état** (ou **automate**) permet de modéliser le comportement d’un **système à événements discrets** (SED). Le comportement à un instant donné d’un SED dépend de son état (ou plus précisément de son état global : voir la notion de configuration d’état dans concepts avancés) qui résulte du comportement passé du système, ET d’une éventuelle transition en cours.

• Un **diagramme états-transitions (statechart)** décrit graphiquement une machine à états.

• La modélisation par une machine d’état du comportement d’un SED s’effectue en décrivant :

* les états possibles du système ;
* les transitions, entre états ou internes aux états, déclenchées par des occurrences d’événements sous certaines conditions ;
* les comportements associés aux états et aux transitions.

• Un tel système est aussi dit **réactif** car il répond à des stimuli internes ou externes.

**Notion d’état (state)**

Un **état** correspond à une situation « durable » du système. Un comportement est associé à un état, ne serait-ce qu’une attente.

**Notion de pseudo-état (pseudostate)**

Un **pseudo-état** correspond à une situation « transitoire » du système.

**Notion de transition (transition)**

Une **transition** indique la possibilité d’évolution d’une situation à une autre. Une transition est déclenchée (ou franchie) sur l’occurrence (l’apparition) d’un événement sous certaines conditions. Un comportement transitoire peut être associé à une transition.

**Notion d’événement (event)**

Un **événement** est un fait qui apparait à un instant donné. Il est supposé instantané.

|  |  |
| --- | --- |
| **Exemple d’un télérupteur**  • Mise sous tension 🡪 activation du pseudo-état initial 🡪 franchissement de T0 🡪 activation de l’état « lampe\_éteinte ».  • Appui sur un bouton poussoir 🡪 occurrence de l’événement « appui\_bp » 🡪  a) si « lampe\_éteinte » actif alors : désactivation de « lampe\_éteinte » 🡪 franchissement de T1 🡪 activation de « lampe\_allumée » |  |

b) si « lampe\_allumée » actif alors : désactivation de « lampe\_allumée » 🡪 franchissement de T2 🡪 activation de « lampe\_éteinte »

• Perte d’alimentation 🡪 occurrence de l’événement « perte\_alim » 🡪 désactivation de la machine d’état. Le plus souvent, la perte d’alimentation n’est pas représentée.

**Pseudo-état initial (initial pseudostate)**

• Un diagramme d’état a un seul **pseudo-état initial** qui cible l’état actif par défaut (d’une région – voir concepts avancés).

• La transition sortante d’un pseudo-état initial peut avoir un comportement, mais ne peut avoir ni déclencheur ni garde. Le pseudo-état initial est transitoire.

**Etat final (final state)**

• Un diagramme d’état peut avoir un **état final**, plusieurs, ou aucun.

• Un état final est (potentiellement) stable, comme tout état.

**Etat simple (simple state) – Comportements (behaviors) associés – Transition interne**

• Un **état** est soit **actif**, soit **inactif**. Un état devient actif dès que le système entre dans l’état, et inactif dès que le système en est sorti (ou l’a quitté).

• Compartiments facultatifs : nom, comportements internes, et transitions internes.

* Comportements (ou effets) internes facultatifs associés :

|  |  |  |  |
| --- | --- | --- | --- |
| *Mot clé* | *Comportement* | *Il est effectué* | *Commentaire* |
| entry/ | d’entrée | - lors de l’entrée dans l’état | De « courte » durée, il ne peut pas être interrompu. |
| do/ | durant  l’état actif | - tant que l’état est actif ou jusqu’à ce qu’il se termine de lui-même,  - après le comportement d’entrée et avant le comportement de sortie | Il est interrompu en cas de sortie de l’état, s’il n’est pas déjà terminé de lui-même. |
| exit/ | de sortie | - lors de la sortie de l’état | De « courte » durée, il ne peut pas être interrompu. |

* Une transition interne s’exécute sans quitter l’état dans lequel elle est définie, ni y rentrer ; les comportements d’entrée et de sortie de l’état ne sont donc pas effectués.

**Transition simple – Comportement associé**

• Une **transition simple** est un arc orienté qui relie un **état** dit **source** à un **état** dit **cible**.

• Une transition simple est dite :

* **externe** si la transition relie extérieurement l’état source à l’état cible ; si l’état source est aussi l’état cible alors on dit parfois que la transition est propre ou réflexive ;
* **interne** si la transition est définie à l’intérieur de l’état (sans en sortir ni y entrer) – voir ci-dessus.

• Une **transition simple** est dite :

* **atteinte**, lorsque l'état source est actif ;
* **traversée** (ou franchie ou exécutée), lorsqu'elle est en cours d'exécution ;
* **terminée**, lorsque l’état cible est actif.

• La durée d’une transition est non définie mais elle peut être considérée nulle.

• La syntaxe d’une transition simple est :

<trigger> [<guard>]/<behavior-expression>

où :

<trigger> est un déclencheur lié à un événement ;

<guard> est une expression booléenne (garde) ;

<behavior-expression> est une expression spécifiant un comportement.

• Un comportement transitoire peut être associé à une transition. Le comportement associé est de « courte » durée et ne peut pas être interrompu.

**Transitions simples entre états simples – Comportements associés**



• Comportement lors de la transition externe quand le système quitte « state\_1 » et entre dans « state\_2 ». Dans l’ordre :

|  |  |  |
| --- | --- | --- |
| *Comportement* | *state\_1* | *state\_2* |
| 1. « doActivity\_behavior\_1 » est arrêté s’il n’était pas déjà terminé de lui-même | actif | inactif |
| 1. « exit\_behavior\_1 » est effectué | actif | inactif |
| 1. « behavior-expression\_A » est effectué | trans. | trans. |
| 1. « entry\_behavior\_2 » est effectué | inactif | actif |
| 1. « doActivity\_behavior\_2 » est débuté | inactif | actif |

• Comportement lors de la transition externe quand le système quitte et rentre dans « state\_2 ». Dans l’ordre :

|  |  |  |
| --- | --- | --- |
| *Comportement* | *state\_1* | *state\_2* |
| 1. « doActivity\_behavior\_2 » est arrêté s’il n’était pas déjà terminé de lui-même | inactif | actif |
| 1. « exit\_behavior\_2 » est effectué | inactif | actif |
| 1. « behavior-expression\_B » est effectué | trans. | trans. |
| 1. « entry\_behavior\_2 » est effectué | inactif | actif |
| 1. « doActivity\_behavior\_2 » est débuté | inactif | actif |

• Comportement lors de la transition interne à « state\_1 » (actif) :

* + « event\_1\_behavior » est effectué ;
  + « entry\_behavior\_1 » et « exit\_behavior\_1 » ne sont pas effectués ;
  + « doActivity\_behavior\_1” n’est pas interrompu.

• Comportements : transition interne vs transition externe « propre » (i.e. état source = état cible)

* Transition externe « propre » : arrêt et redémarrage du comportement durant l’état actif, exécution des comportements d’entrée et de sortie.
* Transition interne : pas d’arrêt et de redémarrage du comportement durant l’état actif, pas d’exécution des comportements d’entrée et de sortie.

**Evénement (event)**

**• Seule l’occurrence (l’apparition) d’un événement peut déclencher une transition.**

• En SysML-UML, le concept d’événement toujours occurrent n’est pas prévu mais il existe le concept d’événement d’achèvement - voir plus loin.

• Types d’événements en SysML-UML :

|  |  |  |
| --- | --- | --- |
| *Evénement* | *Désignation* | *Commentaires* |
| de signal  (signal event) | par son nom | Ex : « appui\_bp » (appui sur bouton poussoir),  « clic\_droit\_souris », « set », « inc », etc. |
| de changement  (change event) | when(boolean\_expression) | Lorsque l’expression passe de faux à vrai  Ex : « when(compteur = 10)  », « when(age>62)  » |
| temporel  (time event) | after(temporal\_duration)  at(date), when(date = date0) | Durée écoulée depuis l’entrée dans un état  Temps absolu |
| d'appel  (call event) | par son nom | réception d’un appel d’opération |

|  |  |
| --- | --- |
| **Exemple d’un télérupteur**  • L’appui sur le bouton poussoir « bp » provoque l’occurrence de l’événement « appui\_bp ». Par suite :  a) si « lampe\_éteinte » actif alors : désactivation de « lampe\_éteinte » 🡪 franchissement de T1 🡪 activation de « lampe\_allumée »  b) si « lampe\_allumée » actif alors : désactivation de « lampe\_allumée » 🡪 franchissement de T2 🡪 activation de « lampe\_éteinte »  • Le relâchement du bouton poussoir, provoque l’occurrence de l’événement « relache\_bp » qui n’est pas utilisé dans la machine d’état donnée précédemment. |  |

**Evénements « front montant » et « front descendant » d’une variable binaire / booléenne**

|  |  |
| --- | --- |
| • L’événement front montant de , noté , correspond au passage de la valeur 0 à la valeur 1 de la variable binaire . Pour une variable booléenne, le front montant correspond au passage de la valeur faux à la valeur vrai.  • Similairement, l’événement front descendant de , noté , correspond au passage de la valeur 1 à la valeur 0 de la variable binaire . Pour une variable booléenne, le front descendant correspond au passage de la valeur vrai à la valeur faux. | ,  : *notations hors SysML* |

• Deux événements sont corrélés s’ils dépendent d’une même variable ou de variables dépendantes.

Exemples : et sont corrélés, si et alors et sont corrélés.

• Formellement, les occurrences de deux événements non corrélés ne se produisent jamais simultanément.

Si a et b sont deux variables indépendantes alors

• Exemples d’événements associés à une combinaison de variables (les variables a, b et c sont indépendantes) :

est différent de On a :

est différent de On a :

• En pratique, pour toute réalisation technologique, les occurrences de deux événements non corrélés peuvent être vues comme simultanées s’il n’y a pas d’impossibilité physique. Dans ce cas, il faut les prioriser.

• Si la variable est discrétisée en temps, elle prendra les valeurs successives : , on peut alors déterminer par :

**Condition de garde (guard)**

• L’expression booléenne de la condition de garde associée à une transition est évaluée seulement lors de l’occurrence de l’événement.

• En l’absence d’expression booléenne explicite, le résultat de la condition de garde est vrai.

• Si le résultat de cette condition est faux alors la transition n’est pas déclenchée et l’occurrence de l’événement est perdue.

**Condition de garde vs événement de changement (when)**

• L’expression booléenne d’une condition de garde est évaluée seulement lors de l’occurrence de l’événement.

• L’expression booléenne d’un événement de changement est, conceptuellement, évaluée en continu.

**Règle d’évolution d’une transition simple entre deux états simples**

• Dans ce cas, une **transition** est dite **validée** (ou **éligible**, ou autorisée, ou permise) si et seulement si :

* son état source est actif ;
* l’événement occurrent (qui apparait) est un déclencheur (trigger) de la transition ;
* le résultat de la condition de garde associée à ce déclencheur est vrai.

• Plusieurs transitions peuvent être validées par la même occurrence d'événement. Ainsi, la validation est une condition nécessaire mais non suffisante pour déclencher (ou franchir) une transition.

• A l’occurrence d’un événement, la machine à états :

1. détermine les transitions validées (voir ci-dessus) ;
2. choisit la transition à déclencher le cas échéant ;
3. arrête le comportement durant l’état actif (do/) de l’état source le cas échéant ;
4. effectue le comportement de sortie (exit/) de l’état source le cas échéant ;
5. désactive l’état source ;
6. effectue le comportement associé à la transition le cas échéant ;
7. active l’état cible ;
8. effectue le comportement d’entrée (entry/) de l’état cible le cas échéant ;
9. démarre le comportement durant l’état actif (do/) de l’état cible le cas échéant ;
10. attend une nouvelle occurrence d’un événement.

• Si aucune transition n’est validée alors l’occurrence de l’événement est perdue.

• Le choix de la transition à déclencher parmi les transitions validées résulte d’un éventuel ordre de priorité ou « du hasard ». Il convient bien sûr d’éviter cette circonstance notamment par le choix des conditions des gardes.

• Une occurrence d’événement permet de déclencher une seule transition. En conséquence, si au moins deux transitions étaient validées alors une seule serait déclenchée. Cela est vrai sauf dans le cas de régions orthogonales (voir concepts avancés plus loin) où une occurrence permet de déclencher une transition par région orthogonale.

|  |  |
| --- | --- |
| **Exemple des modes d’une horloge**    • Le poussoir « set » permet de parcourir les états d’affichage de l’heure « show\_time », du réglage de l’heure « set\_hours » et des minutes « set\_minutes ».  • Le poussoir « inc » permet de régler par incrémentation les heures ou les minutes.  • Supposons l’état « set\_hours » actif. A l’occurrence de l’événement « set », seule la transition T2 est validée et donc déclenchée. Par suite : « show\_hours » est arrêté 🡪 l’état « set\_hours » est désactivé 🡪 l’état « set\_minutes » est activé 🡪 « beep » est effectué 🡪 « show\_minutes » est démarré. |  |

• Supposons encore l’état « set\_hours » actif. A l’occurrence de l’événement « inc », seule la transition T4 est validée et donc déclenchée. Par suite : « show\_hours » est arrêté 🡪 l’état « set\_hours » est désactivé 🡪 la variables « hours » est incrémentée modulo 24 🡪 l’état « set\_hours » est réactivé 🡪 « beep » est effectué 🡪 « show\_hours » est redémarré. Si la transition T4 était interne plutôt qu’externe à « set\_hours », « beep » ne serait pas effectué.

**Transition d’achèvement (completion transition) sortant d’un état simple source - Evénement d’achèvement**

• Une transition d’achèvement est dépourvue de déclencheur explicite ; elle peut avoir une condition de garde.

• Un événement d’achèvement est implicitement associé à une transition d’achèvement et peut la déclencher.

• Un événement d’achèvement apparait automatiquement à la fin des comportements associés à l’état source d’une transition d’achèvement, c’est-à-dire, après la fin du comportement d’entrée « entry\_behavior » s’il est défini, et après la fin du comportement durant l’état actif « doActivity\_behavior » s’il est défini – ce qui nécessite qu’il se termine de lui-même. Si aucun de ces deux types de comportements n'est associé à l’état source, l'événement d'achèvement est généré dès l'entrée dans l'état source.



**Pseudo-état de jonction (jonction pseudostate)**

• Un pseudo-état de jonction réalise une branche conditionnelle statique : toutes les gardes sont évaluées avant le début d’une transition, et toutes les gardes le long d’un chemin doivent être évaluées à vrai avant le déclenchement de la transition correspondante.

• Il possède un ou plusieurs segments entrants et un ou plusieurs segments sortants.

• Seuls les segments entrants peuvent être étiquetés par des événements.

• Si les conditions de garde associées à plusieurs segments sortants peuvent prendre en même temps la valeur vrai, alors les transitions peuvent être en conflit. Il est très conseillé de les rendre exclusives.

• Une garde prédéfinie notée « else » peut être définie pour au plus une transition sortante.

• Une jonction est un pur artifice graphique permettant de partager (on dit aussi de « factoriser ») des segments de transition afin d’améliorer la lisibilité. Exemple : les deux diagrammes ci-dessous sont équivalents.

|  |  |
| --- | --- |
|  |  |

**Pseudo-état de choix** (choice pseudostate)

• Un pseudo-état de choix réalise une branche conditionnelle dynamique : les gardes situées après le pseudo-état de choix sont évaluées au moment où il est atteint, contrairement à une jonction. Ainsi, la décision sur le chemin à emprunter peut être fonction des comportements effectués sur le chemin avant le pseudo-état de choix.

• Il possède un seul segment entrant et deux ou plusieurs segments sortants.

• Si aucune garde des segments sortants n’est évaluée à vrai, alors le modèle est considéré comme mal formé. Pour éviter cela, il est conseillé d’utiliser une transition sortante avec la garde « else » prédéfinie. Exemple :



**B) Concepts avancés : état composite, configuration d’état, …**

**Etat composite (composite state) dit aussi « super-état »**

• L'utilisation d'états composites permet de hiérarchiser et structurer une description.

• Un état composite, est soit un **état composite « simple »** c’est-à-dire avec une unique région, soit un **état composite « orthogonal »** c’est-à-dire avec plusieurs régions orthogonales.

• Chaque région contient un ou plusieurs états (simples ou composites).

• Deux régions sont orthogonales l'une à l'autre si elles appartiennent au même état. Les régions orthogonales entre-elles sont simultanément actives ou inactives. Elles concourent ensemble au comportement du système.

• Aucune transition entre régions orthogonales n’est autorisée. Cependant, une contrainte de garde peut impliquer des tests d'états d’autres régions.

Par exemple, [in region1::state2] ou [not in regionB::state4]).

• Un état composite, est décrit par les compartiments : nom, comportements internes et transitions internes comme pour un état simple, auxquels s’ajoute le compartiment de décomposition. Les lignes pointillées divisent le compartiment de décomposition en régions orthogonales.

• Un état composite peut être affiché avec une décomposition cachée afin de simplifier la lecture.

• L’entrée dans un état composite provoque l’activation de toutes ses régions.

• L’entrée dans un état composite à une seule région peut se faire de différentes façons.

* Si une transition entrante se termine sur le bord extérieur de l'état composite, alors l’entrée s’effectue par défaut en activant le pseudo-état initial de sa région.
* Si une transition entrante se termine directement sur un état contenu dans la région de l'état composite, alors l’entrée s’effectue explicitement en activant cet état et tous les états qui le contiennent.
* L’entrée peut aussi s’effectuer via un pseudo-état historique (voir plus loin) ou via un pseudo-état de point d'entrée (voir plus loin).

• L’entrée dans un état composite orthogonal (i.e. avec plusieurs régions) équivaut à une entrée dans chacune de ses régions.

* Si une transition se termine sur le bord extérieur de l'état composite, alors l’entrée dans chacune de ses régions s’effectue par défaut.
* Si une transition entrante se termine directement sur un état contenu dans une région ou plusieurs (dans le cas d’un pseudo-état fourche – voir plus loin), alors l’entrée s’effectue explicitement pour ces régions et par défaut pour toutes les autres régions.

• L’entrée dans un état composite et les comportements associés sont effectués séquentiellement du niveau hiérarchique le plus élevé (le moins profond) – l’état composite lui-même – au niveau le plus faible (le plus profond) de chaque région.

• La sortie d’un état composite provoque la désactivation de toutes ses régions et donc de tous ses états actifs.

• La sortie d'un état composite et les comportements associés sont effectués séquentiellement du niveau hiérarchique le plus faible (le plus profond) au niveau le plus élevé (le moins profond) de chaque région, pour finir par la sortie de l’état composite lui-même.

**Exemple d’un état composite avec une seule région – Décomposition hiérarchique**

• Les deux diagrammes ci-dessous sont fonctionnellement équivalents.

|  |  |
| --- | --- |
|  |  |

• En supposant l’état « state\_1 » actif, le déclenchement de la transition T1 (via la transition T5 pour le diagramme de départ, ou directement pour le diagramme équivalent) entraine séquentiellement :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Comportement* | *state\_1* | *state\_2* | *state\_21* | *state\_22* |
| 1. « doA\_beh\_1 » est arrêté si pas déjà terminé de lui-même | actif | inactif | inactif | inactif |
| 1. « exi\_beh\_1 » est effectué | actif | inactif | inactif | inactif |
| 1. « bA » est effectué | trans. | trans. | trans. | trans. |
| 1. « ent\_beh\_2 » est effectué | inactif | actif | inactif | inactif |
| 1. « doA\_beh\_2 » est démarré | inactif | actif | inactif | inactif |
| 1. « ent\_beh\_21 » est effectué | inactif | actif | actif | inactif |
| 1. « doA\_beh\_21 » est démarré | inactif | actif | actif | inactif |

• En supposant l’état « state\_22 » actif, le déclenchement de la transition T2 pour le diagramme de départ, ou T6 pour le diagramme équivalent entraine séquentiellement :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Comportement* | *state\_1* | *state\_2* | *state\_21* | *state\_22* |
| 1. « doA\_beh\_22 » est arrêté si pas déjà terminé de lui-même | inactif | actif | inactif | actif |
| 1. « exi\_beh\_22 » est effectué | inactif | actif | inactif | actif |
| 1. « doA\_beh\_2 » est arrêté si pas déjà terminé de lui-même | inactif | actif | inactif | inactif |
| 1. « exi\_beh\_2 » est effectué | inactif | actif | inactif | inactif |
| 1. « bB » est effectué | trans. | trans. | trans. | trans. |
| 1. « ent\_beh\_1 » est effectué | actif | inactif | inactif | inactif |
| 1. « doA\_beh\_1 » est démarré | actif | inactif | inactif | inactif |

|  |  |
| --- | --- |
| • Le diagramme ci-contre reprend le diagramme de départ ci-dessus en cachant la région. |  |
| • Le passage du diagramme précédent à un diagramme constitué uniquement d’états simples représenté ci-contre nécessite des précautions :  - l’ordre d’exécution des comportements d’entrée nécessite de respecter celui imposé par la hiérarchie d’un état composite ; idem pour l’ordre d’exécution des comportements de sortie ; ce qui est a priori possible.  - par contre, le comportement « doA\_beh\_2 », est obligatoirement interrompu puis redémarré lors de la transition de « state\_21 » à « state\_22 » (et de « state\_22 » à « state\_21 ») avec le diagramme ci-contre, alors qu’il n’est démarré qu’une fois avec l’état composite « state\_2 » lors de son activation ; ce qui peut être très différent. |  | |

**Exemple d’un état composite avec deux régions orthogonales**

• Le diagramme ci-dessous sert à illustrer plusieurs concepts ci-après.



**Exemple de transition arrivant sur le bord extérieur de l'état composite**

• En supposant l’état « state\_1 » actif, le déclenchement de la transition T1 provoque : la désactivation de l’état actif « state\_1 », le franchissement de la transition T1, l’activation de l’état composite « state\_2 », le franchissement des transitions T10 et T20 (issues des pseudo états initiaux), puis l’activation des états « state\_211 » et « state\_221 » (l’ordre d’activation de « state\_211 » et « state\_221 » n’est pas défini).

• Lors de l’entrée dans l’état composite, les comportements associés sont effectués séquentiellement du niveau hiérarchique le plus élevé au niveau le plus faible de chaque région.

**Exemple de transition partant du bord extérieur de l'état composite**

• En supposant l’état « state\_2 » actif, le déclenchement de la transition T6 provoque : la désactivation des états actifs des régions 1 et 2 (l’ordre de désactivation entre régions n’est pas défini), la désactivation de l’état « state\_2 », le franchissement de la transition T6, puis l’activation de l’état « state\_4 ».

• Lors de la sortie de l’état composite, les comportements associés sont effectués séquentiellement du niveau hiérarchique le plus faible au niveau le plus élevé de chaque région.

**Pseudo-état fourche (fork pseudostate)**

• Un pseudo-état fourche divise une transition entrante en plusieurs transitions vers des régions orthogonales d’un état composite.

• Les transitions qui sortent d’une fourche ne peuvent pas avoir de garde ou de déclencheur.

• Exemple : en supposant l’état « state\_1 » actif, le déclenchement de la transition T2 provoque dans l’ordre : la désactivation de l’état « state\_1 », le franchissement des transitions T2, T2R1 et T2R2, l’activation de l’état composite « state\_2 », puis l’activation des états « state\_211 » et « state\_222 » (l’ordre d’activation n’est pas défini).

**Pseudo-état jointure (join pseudostate)**

• Un pseudo-état jointure fusionne plusieurs transitions provenant de régions orthogonales.

• Les transitions qui se terminent sur une jointure ne peuvent pas avoir de garde ou de déclencheur.

• Les pseudo-états de jointure effectuent une synchronisation, dans laquelle toutes les transitions entrantes doivent se terminer avant que l'exécution puisse se poursuivre via une transition sortante.

• Exemple : en supposant les états « state\_212 » et « state\_222 » actifs, le déclenchement de la transition T5 provoque dans l’ordre : la désactivation des états « state\_212 » et « state\_222 » (l’ordre de désactivation n’est pas défini), la désactivation de l’état « state\_2 », le franchissement des transitions T5R1, T5R2 et T5 puis l’activation de l’état « state\_4 ».

**Transition d’achèvement (completion transition) d’un état composite**

• Pour un état composite, un événement d'achèvement est généré si :

* toutes les activités internes (exit/, do/, …) ont terminé leur exécution, et
* toutes ses régions orthogonales ont atteint un état final.

• Exemple : lors de l’activation des états finaux des régions 1 et 2 de l’état composite, un événement d’achèvement est généré. Si le résultat de la condition de garde [gD] est vrai alors la transition d’achèvement T4 est déclenchée provoquant : la désactivation des états finaux des régions 1 et 2, la désactivation de l’état composite « state\_2 », le franchissement de la transition T4, puis l’activation de l’état « state\_3 ». L’association des états finaux et de la transition d’achèvement réalisent une synchronisation. Mais attention, l’évènement d’achèvement peut être perdu si le résultat de la condition de garde [gD] est faux lors de son occurrence.

**Configuration d’état (state configuration) ou « état global » – Configuration d’état actif**

• Une **configuration d'état** (d'un état composite ou d'une machine à états) est un ensemble d’états hiérarchisés acceptable du diagramme.

• Une **configuration d'état actif** est l’ensemble des états hiérarchisés actifs à un moment de l’exécution de la machine à états.

• Réciproquement, un état est actif s'il fait partie de la configuration d'état actif.

• Une machine à états en cours d'exécution ne peut être que dans exactement une configuration d'état actif à la fois. Une machine à états évolue donc d’une configuration d’état actif à une autre en réponse à des occurrences d'événements.

• Une **configuration d'état actif** est dite **stable** lorsque :

* aucune autre transition de cette configuration n'est validée et,
* tous les éventuels comportements d'entrée de cette configuration sont terminés (les éventuels comportements en cours d’état actifs (do/) de cette configuration peuvent continuer à s’exécuter).

• Comme les états peuvent être hiérarchiques et qu'il peut y avoir des comportements associés à la fois aux transitions et aux états, l’entrée du système dans une configuration d'état hiérarchique implique un processus dynamique qui ne se termine qu'après qu'une configuration d'état stable soit atteinte.

**Exemple d’un état composite avec deux régions orthogonales – Configuration d’état**

• Afin de ne pas alourdir, les transitions dans l’exemple ci-dessous ne sont désignées que par leur nom. Configurations d’état actif : {s\_1}, {s\_2, s\_2::s\_211, s\_2::s\_221}, {s\_2, s\_2::s\_211, s\_2::s\_222}, {s\_2, s\_2::s\_221, s\_2::s\_211}, {s\_2, s\_2::s\_212, s\_2::s\_222}, {s\_3}.



• Le graphe ci-dessous représente toutes les configurations d’état actif accessibles (en supposant qu’aucune contrainte particulière n’interdise le franchissement d’une transition) et les transitions y conduisant. Ce graphe est désigné ici par diagramme des configurations d’état actif.



**Pseudo-état historique peu profond (shallow history pseudostate), ou superficiel ou plat**

• Un pseudo-état historique peu profond trace l’état actif le plus récent de sa région (mais pas les éventuels états actifs internes à cet état).

• Une transition entrant dans le pseudo-état historique peu profond équivaut à une transition entrant dans le dernier état actif tracé par l’historique ; le comportement d’entrée de cet état est effectué.

• Une transition (optionnelle) du pseudo-état historique peu profond vers un état par défaut permet de préciser le cas où l’état composite n’a jamais été actif auparavant. En l’absence de cette transition, c’est l’état actif par défaut de la région qui est activé.

**Pseudo-état historique profond (deep history pseudo state)**

• Un pseudo-état historique profond trace la configuration d’état actif la plus récente de sa région.

• Une transition entrant dans le pseudo-état historique profond équivaut à une transition entrant dans les états les plus internes de la dernière configuration d’état actif tracée par l’historique ; les comportements d’entrée rencontrés en parcourant les chemins de restauration de la configuration d’état actif sont effectués.

• Une transition (optionnelle) du pseudo-état historique profond vers un état par défaut permet de préciser le cas où l’état composite n’a jamais été actif auparavant.

|  |  |
| --- | --- |
| **Exemple d’un pseudo-état historique profond : lavage automatique de voiture**  • Lors du lavage ou du séchage, l’arrêt d’urgence provoque une mise en d’attente. Si la reprise est effectuée dans un délai inférieur à 2 minutes, le lavage ou le séchage (états composites) est repris dans le dernier sous-état actif tracé par l’historique profond. Si un état historique peu profond avait été utilisé, c’est toute la séquence du lavage ou du séchage qui aurait été recommencée.  • Lors du lustrage, l’arrêt d’urgence provoque un arrêt final. |  |

**Sommet (vertex / vertices)**

• Un **sommet** désigne différents types de nœuds du graphe : états, pseudo-états, points de connexion.

• D’une façon générale (à quelques exceptions près), un sommet est la **source** ou la **cible** d’une ou plusieurs transitions (entrantes ou sortantes).

**Etats composites encapsulés (encapsulated composite states)**

• L’encapsulation d’un état composite permet d’améliorer la lisibilité et la structuration d’un diagramme en évitant toute liaison directe entre transitions externes et sommets internes de l’état composite.

• Les liaisons entre transitions externes et sommets internes de l’état composite s’effectuent alors au moyen de pseudo-états dits de points d'entrée et de sortie.

**Pseudo-état de point d’entrée (entry point pseudostate) d’un état composite**

• Les points d'entrée constituent les points de terminaison pour les transitions entrantes et les points d'origine pour les transitions qui se terminent sur des sommets internes de l'état composite. Ils sont placés sur la bordure et représentés par des cercles vides.

• L'exécution du comportement d'entrée de l'état composite (s'il est défini) se produit entre le comportement de la transition entrante et le comportement de la transition sortante.

• S'il n'y a pas de transition sortante à l'intérieur de l'état composite, alors la transition entrante effectue simplement une entrée d'état par défaut.

• Si plusieurs régions sont impliquées, le point d'entrée agit comme un pseudo-état de fourche.

**Pseudo-état de point de sortie (exit point pseudostate) d’un état composite**

• Les points de sortie constituent les points de terminaison pour les transitions issues de sommets internes de l'état composite et les points d'origine pour les transitions sortantes. Ils sont placés sur la bordure et représentés par des cercles avec une croix.

• Si l'état composite a un comportement de sortie défini, il est exécuté après tout comportement de la transition interne entrante et avant tout comportement de la transition externe sortante.

• Si plusieurs transitions de régions orthogonales au sein de l'état se terminent sur ce pseudo-état, alors il agit comme un pseudo-état de jointure.

**C) Tableau des symboles graphiques**

|  |  |
| --- | --- |
|  | **Etat simple (simple state) :** un état est soit actif, soit inactif.  • Un état devient actif dès que le système entre dans l’état, et inactif dès que le système en est sorti (ou l’a quitté)  • Un état simple est décrit par les compartiments facultatifs : nom, comportements internes, et transitions internes.  • Comportement (ou effet) interne :   * entry/ : effectué lors de l’entrée dans l’état ; * do/ : effectué durant l’état actif (après les comportements d’entrée et avant les comportements de sortie), effectué tant que l’état est actif ou jusqu’à ce qu’il se termine par lui même ; * exit/ : effectué lors de la sortie de l’état.   Les comportements d’entrée « entry\_behavior » et de sortie « exit\_behavior » sont de « courtes » durées et ne peuvent pas être interrompus.  Les comportements non terminés durant l’état actif « doActivity\_behavior » sont interrompus en cas de sortie de l’état.  • Transition interne (écriture habituelle d’une transition). Une transition interne s’exécute sans quitter l’état dans lequel elle est définie, ni y rentrer ; les comportements d’entrée et de sortie ne sont donc pas effectués. |
|  | **Etat composite (composite state) :** un état composite, dit aussi « super-état », est décrit par les compartiments : nom, comportements internes, transitions internes, auxquels s’ajoute le compartiment de décomposition.  • Les lignes pointillées divisent le compartiment de décomposition en **régions** dites **orthogonales** qui concourent ensemble au comportement. Elles sont aussi dites concurrentes.  • Un état composite est un état contenant une ou plusieurs régions, et chaque région contient un graphe éventuellement hiérarchique.  • L’entrée dans un état composite provoque l’activation de toutes les régions (après avoir effectué le comportement d’entrée).  • L’entrée dans un état composite et les comportements associés sont effectués séquentiellement du niveau hiérarchique le plus élevé au niveau le plus faible de chaque région.  • La sortie de l’état composite provoque la désactivation de toutes les régions et de tous les états actifs.  • La sortie d'un état composite et les comportements associés sont effectués séquentiellement du niveau hiérarchique le plus faible au niveau le plus élevé de chaque région, pour finir par la sortie de l’état composite lui-même.  • Un état composite n’est pas autorisé à avoir à la fois des régions et une sous-machine. |
|  | **Etat composite (composite state) :** un état composite peut être affiché avec une décomposition cachée. |
|  | **Etat final (final state) :** un diagramme d’état peut avoir un état final, plusieurs, ou aucun.  • Selon son emplacement, l’état final marque la fin d’activation d’un diagramme d’états-transitions, d’un état-composite, ou d’une région de manière la plus générale.  • L’état final est (potentiellement) stable, comme tout état. |
|  | **Pseudo-état initial (initial pseudo state) :** un diagramme d’état a un seul pseudo-état initial qui cible l’état actif par défaut d’une région.  • La transition sortante du pseudo-état initial peut avoir un comportement, mais pas de déclencheur ou de garde.  • Contrairement à l’état final, le pseudo-état initial est transitoire. |
|  | **Pseudo-état de jonction (junction pseudo state) :** un pseudo-état de jonction est un artifice graphique permettant de partager (factoriser) des segments de transition afin d’améliorer la lisibilité.  • Une jonction réalise une branche conditionnelle statique : toutes les conditions de garde sont évaluées avant l'exécution de toute transition composée contenant ce pseudo-état.  • Toutes les gardes le long d’un chemin doivent être évaluées à vrai avant le déclenchement de la transition correspondante.  • Si plus d'une transition composée est validée (plusieurs chemins possibles), l'une d'entre elle est choisie mais le choix n'est pas défini. Il est très conseillé de les rendre exclusives.  • Une garde prédéfinie notée « else » peut être définie à au plus une transition sortante.  • Seuls les segments entrants peuvent contenir des déclencheurs. |
|  | **Pseudo-état de choix (choice pseudo state) :** un pseudo-état de choix permet d’introduire une alternative (voir aussi flux de contrôle ci-dessous).  • Le pseudo-état de choix réalise une branche conditionnelle dynamique : les gardes situées après le pseudo-état de choix sont évaluées au moment où il est atteint, contrairement à une jonction. Ainsi, la décision sur le chemin à emprunter peut être fonction des comportements effectués sur le chemin avant le pseudo-état de choix.  • Si plus d'une garde est évaluée à vrai, l'une des transitions correspondantes est sélectionnée, mais le choix n'est pas défini.  • Si aucune garde des segments sortants n’est évaluée à vrai, alors le modèle est considéré comme mal formé. Pour éviter cela, il est conseillé d’utiliser une transition sortante avec la garde « else » prédéfinie.  • Il possède un seul segment entrant et deux ou plusieurs segments sortants. |
|  | **Pseudo-état fourche (fork pseudostate) :** un pseudo-état fourche divise une transition entrante en plusieurs transitions sortantes se terminant sur des sommets des régions orthogonales d’un état composite.  • Les transitions sortant d’une fourche ne peuvent pas avoir de garde ou de déclencheur.  **Pseudo-état jointure (join pseudostate) :** un pseudo-état jointure fusionne plusieurs transitions provenant de sommets de régions orthogonales ; il effectue une synchronisation.  • Les transitions entrant sur une jointure ne peuvent pas avoir de garde ou de déclencheur.  • Toutes les transitions entrantes doivent se terminer avant que l'exécution puisse se poursuivre via une transition sortante. |
|  | **Pseudo-état historique peu profond (shallow history pseudo state), ou superficiel :** un pseudo-état historique peu profond trace l’état actif le plus récent de sa région (mais pas les éventuels états actifs internes à cet état).  • Une transition entrant dans le pseudo-état historique peu profond équivaut à une transition entrant dans le dernier état actif tracé par l’historique ; le comportement d’entrée de cet état est effectué.  • Une transition (optionnelle) du pseudo-état historique peu profond vers un état par défaut permet de préciser le cas où l’état composite n’a jamais été actif auparavant. En l’absence de cette transition, c’est l’état actif par défaut de la région qui est activé. |
|  | **Pseudo-état historique profond (deep history pseudo state) :** un pseudo-état historique profond trace la configuration d’état actif la plus récente de sa région.  • Une transition entrant dans le pseudo-état historique profond équivaut à une transition entrant dans les états les plus internes de la dernière configuration d’état actif tracée par l’historique ; les comportements d’entrée rencontrés en parcourant les chemins de restauration de la configuration d’état actif sont effectués.  • Une transition (optionnelle) du pseudo-état historique profond vers un état par défaut permet de préciser le cas où l’état composite n’a jamais été actif auparavant. |
|  | **Pseudo-état de point d’entrée (entry point pseudostate) :** le pseudo-état de point d’entrée représente un point d’entrée d’une machine à états encapsulée ou d’un état composite encapsulé. Il est placé sur la bordure (frontière).  • Il y a au plus une transition du point d'entrée vers un sommet dans chaque région de la machine à états ou de l'état composite possédant le point d'entrée.  • Si plusieurs régions sont impliquées, le point d'entrée agit comme un pseudo-état de fourche.  • Si l'état contenant le pseudo-état d’entrée a un comportement d'entrée associé, ce comportement est exécuté avant tout comportement associé à la transition sortante de ce pseudo-état.  **Pseudo-état de point de sortie (exit point pseudostate) :** le pseudo-état de point de sortie est un point de sortie d’une machine à états encapsulée ou d’un état composite encapsulé. Il est placé sur la bordure (frontière).  • Les transitions se terminant sur un point de sortie dans n'importe quelle région de l'état composite ou d'une machine à états implique la sortie de cet état composite ou état sous-machine (avec exécution de son comportement de sortie associé).  • Si plusieurs transitions de régions orthogonales au sein de l'état se terminent sur ce pseudo-état, alors il agit comme un pseudo-état de jointure. |
|  | **Pseudo-état de terminaison (terminate pseudostate) :** le pseudo-état de terminaison marque la fin de l’exécution d’une machine à états.  • L’exécution de la machine à états prend fin immédiatement lors de l’entrée dans un pseudo-état de terminaison.  • La machine à états ne quitte aucun état et n'effectue aucun comportement de sortie. Tout comportement en cours d'exécution est automatiquement abandonné. |
|  | **Flux de contrôle :** une représentation graphique des transitions, et des transitions composées, existe. Elle est utile si le comportement peut être décrit par une séquence basée sur un flux de contrôle.  Cette notation se présente sous la forme d’un graphe orienté, qui consiste en un ou plus de symboles graphiques interconnectés par des arcs orientés qui représentent le flux de contrôle.  Bien que cette notation contienne des éléments graphiques rappelant la notation utilisée pour les activités, il s’agit d’une forme distincte applicable uniquement aux machines à états.  **Symbole du point de choix (choice point symbol) :** ce symbole correspond directement au pseudo-état de choix (choice pseudo state) et utilise la même notation (voir ci-dessus).  **Symbole de fusion (merge symbol) :** un symbole de fusion utilisé pour joindre plusieurs arcs de flux de contrôle correspond directement à un pseudo-état de jonction et utilise la même notation (voir ci-dessus). |
|  | **Symbole d’action (action symbol) :** participe à spécifier le comportement dans la branche. |
|  | **Symbole de réception du signal (receive signal symbol) :** le symbole de réception du signal correspond au déclencheur de la transition.  Déclencheur et garde sont contenus dans le symbole :  <trigger> [‘,’ <trigger>]\* [‘[‘ <guard> ‘]’]  Seuls les types d’événement signal et change sont autorisés pour les déclencheurs. |
|  | **Symbole d’envoi d’un signal (signal send symbol) :** le symbole d’envoi d’un signal participe à spécifier le comportement dans la branche de la transition par une action spéciale d’envoi d’un signal. |
|  | **Transition :** une transition est un arc orienté qui relie un sommet dit source à un sommet dit cible.  • Une transition est définie par l’expression BNF :  [<trigger> [‘,’ <trigger>]\* [‘[‘ <guard>’]’] [‘/’ <behavior-expression>]]  où :  <trigger> est un déclencheur lié à un événement ;  <guard> est une expression booléenne (garde) ;  <behavior-expression> est une expression spécifiant un comportement.  • Une transition est dite :  - atteinte, lorsque l'état source est actif ;  - traversée (ou franchie ou exécutée) lorsqu'elle est en cours ;  - terminée, lorsque l’état cible est actif.  **Types d’événements SysML :**  - signal (signal event) : désigné par son nom  - changement (change event) : when(boolean\_expression)  - temporel (time event) : after(temporal\_duration), at(date),  when(date = date0)  - d'appel (call event) : appel d’une opération désigné par son nom  **Condition de garde (guard)**  • L’expression booléenne de la condition de garde associée à une transition est évaluée seulement lors de l’occurrence de l’événement.  • En l’absence d’expression booléenne explicite, le résultat de la condition de garde est vrai.  • Si le résultat de cette condition est faux alors la transition n’est pas déclenchée et l’occurrence de l’événement est perdue.  • Dans une transition simple, la garde est évaluée avant le déclenchement de la transition.  • Dans les transitions composées impliquant plusieurs gardes, toutes les gardes sont évaluées avant qu’une transition ne soit déclenchée, à moins qu’il y ait des points de choix le long d’un ou plusieurs des chemins.  • La syntaxe d’une condition de garde vérifiant si un état nommé state0 est actif est : [in state0]. Celle vérifiant qu’il n’est pas actif est : [not in state0].  **Comportement (behavior)**  • Le comportement associé à la transition est effectué lors de l’exécution de la transition. |
|  | **Transition d’achèvement (completion transition) :** une transition d’achèvement est dépourvue de déclencheur explicite.  • Un événement d’achèvement est implicitement associé à une transition d’achèvement et peut la déclencher.  • Pour un état simple, un événement d’achèvement apparait automatiquement à la fin des comportements associés à l’état source d’une transition d’achèvement.  • Pour un état composite, un événement d'achèvement est généré si toutes les activités internes (exit/, do/, …) ont terminé leur exécution, et toutes ses régions orthogonales ont atteint un état final. |

**D) Concepts avancés : généralisation et compléments**

**Machine d’état (state machine)**

• Le **diagramme états-transitions** d’une machine d’état comprend une ou plusieurs régions.

• Chaque **région** contient un graphe éventuellement hiérarchique.

• Chaque **graphe** comprend un ensemble de **sommets** interconnectés par des **arcs** représentant des **transitions**.

**Région (region)**

• Une région décrit une partie du comportement de la machine (tout le comportement s’il n’y a qu’une région).

• Une région devient active lorsque l’état la contenant devient actif ou, si elle se situe au niveau le plus élevé de la machine, lorsque la machine d'état la contenant commence à s'exécuter.

• L’activation d’une région peut être implicite par exemple lorsqu’une transition externe se termine sur l'état la contenant, ou explicite lorsqu'une transition se termine sur l'un des sommets contenus de la région.

• Deux régions ou plus sont orthogonales l'une à l'autre si elles appartiennent au même état ou, au niveau le plus élevé, à la même machine à états.

• Les régions orthogonales entre-elles sont simultanément actives. Lorsqu'une région d'un état orthogonal est activée explicitement, cela entraîne l'activation par défaut de toutes ses régions orthogonales, à moins que ces régions ne soient également entrées explicitement (plusieurs régions orthogonales peuvent être entrées explicitement en parallèle via des transitions provenant du même pseudo-état de fourche).

**Etat (state)**

• Les types d'états suivants sont distingués :

* Un état simple (simple state), ou état feuille (leaf state), n'a ni sommet ni transition interne.
* Un état composite (composite state), est soit un état composite « simple » avec exactement une région, soit un état composite « orthogonal » avec plusieurs régions orthogonales. Un état inclus dans une région d'un état composite est appelé un « sous-état » de cet état composite. On désigne aussi parfois par « super-état » un état composite.
* Un état sous-machine (submachine state) fait référence à une machine à états entière incluse dans l'état.

**Etat sous-machine (submachine state) – Sous-machine**

• Une sous-machine (à états) est une machine à états spécifiée une fois et réutilisable plusieurs fois, comme une macro dans les langages de programmation.

• Chaque état sous-machine représente une instanciation distincte d'une sous-machine.

• Un état sous-machine est sémantiquement équivalent à un état composite. Les régions de la sous-machine sont les régions de l'état composite. Les comportements d'entrée (entry), de sortie (exit), durant l’état (do), et les transitions internes sont définis dans l'état sous-machine.

• Une sous-machine peut être entrée via son pseudo-état par défaut (initial) ou via l'un de ses points d'entrée. Cela peut impliquer l'entrée dans un état composite non orthogonal ou orthogonal avec des régions.

• L'entrée via le pseudo-état initial a la même signification que pour les états composites ordinaires.

• Un point d'entrée équivaut à un pseudo-état de jonction (une fourche dans les cas où l'état composite est orthogonal). Cela implique que le comportement d'entrée de l'état composite est exécuté, suivi de la transition du point d'entrée au sommet cible dans l'état composite.

• Toutes les gardes associées à ces transitions de point d'entrée doivent être évaluées à vrai pour que la spécification soit bien formée.

• Une sous-machine peut être quittée à la suite de : l’atteinte de son état final, le déclenchement d'une transition de groupe provenant d'un état de sous-machine, ou via l'un de ses points de sortie. La sortie via un état final ou par une transition de groupe a la même signification que pour les états composites ordinaires.

**Référence de point de connexion (ConnectionPointReference)**

• Une référence de point de connexion représente une utilisation (dans le cadre d'un état de sous-machine) d'un point d'entrée / sortie défini dans la sous-machine d'état référencée par l'état sous-machine.

• Les références de points de connexion sont des sources / cibles de transitions impliquant des sorties / entrées dans la sous-machine référencée par un état sous-machine.

• Les régions de la sous-machine sont entrées par les pseudo-états de point d'entrée correspondants.

• Lorsqu'une région de la sous-machine atteint le point de sortie correspondant, l'état de la sous-machine est quitté via ce point de sortie.

**Transition (transition) – Déclencheur (trigger)**

• Une **transition** est un arc orienté unique provenant d'un seul sommet source et se terminant sur un seul sommet cible, qui spécifie un fragment valide du comportement d'une machine à états.

• La source et la cible d’une transition peuvent être le même sommet.

• Une transition peut avoir un comportement associé qui est exécuté lorsque la transition est traversée (exécutée).

• La durée d'un parcours de transition n'est pas définie, ce qui permet différentes interprétations sémantiques, y compris le temps « zéro » et non « zéro ».

• L’exécution d'une **transition composée** (voir ci-après) fait passer une machine d'état d'une configuration d'état stable à une autre.

• Une **transition** est dite :

* **atteinte**, lorsque l'exécution de sa machine d'état a atteint son sommet source (c'est-à-dire que son état source est dans la configuration d'état actif) ;
* **traversée** (ou exécutée), lorsqu'elle est en cours d'exécution (avec tout comportement associé) ;
* **terminée**, après avoir atteint son sommet cible.

• Une transition peut posséder un ensemble de **déclencheurs** (triggers), dont chacun spécifie un événement dont l'occurrence, une fois distribuée, peut déclencher le parcours de la transition.

• Un **déclencheur de transition** est dit **validé** si l'occurrence d'événement distribuée correspond à son type d'événement.

• Lorsque plusieurs déclencheurs sont définis pour une transition, ils sont logiquement disjonctifs, c'est-à-dire que si l'un d'entre eux est validé, la transition sera déclenchée.

**Evénement différé (deferred event)**

• Un événement peut être différé par un état (simple, composite ou sous-machine). L’événement est alors différé tant que cet état est dans la configuration active.

• Les occurrences de cet événement restent dans le pool d'événements jusqu'à ce qu'une configuration d'état soit atteinte où cet événement n’est plus différé.

• Si un événement différé est utilisé explicitement dans un déclencheur d'une transition dont la source est l'état qui diffère cet événement, alors l'occurrence de cet événement est acceptée ; l’événement n'est pas différé dans ce cas.

• Similairement, une occurrence d’un événement différé peut être acceptée par l’exécution d’une activité d’état actif (do/).

**Types de transition par rapport à la source (transition kinds relative to source)**

• Une **transition externe** (external) sort de son sommet source. Si le sommet est un état, l'exécution de cette transition entraine l'exécution de tout comportement de sortie associé à cet état.

• Une **transition locale** (local) ne sort pas de l’état la contenant et, par conséquent, le comportement de sortie de l'état la contenant n’est pas exécuté. Cependant, pour une transition locale, le sommet cible doit être différent du sommet source. Une transition locale ne peut exister que dans un état composite.

• Une **transition interne** (internal) est un cas particulier de transition locale qui est une auto-transition (c'est-à-dire avec les mêmes états source et cible), de sorte que l'état n'est jamais sorti (et donc pas rentrée), ce qui signifie qu'aucun comportement de sortie ou d'entrée n'est exécuté lorsque cette transition est exécutée. Ce type de transition ne peut être défini que si le sommet source est un état.

**Transition de haut niveau ou de groupe (high-level / group transition)**

• Les transitions dont le sommet source est un état composite sont appelées **transitions de haut niveau ou de groupe**. Si elles sont externes, les transitions de groupe entraînent la sortie de tous les sous-états de l'état composite, exécutant tous les comportements de sortie définis en commençant par les états les plus internes dans la configuration d'état actif.

• En cas de transitions locales, les comportements de sortie de l'état source et les comportements d'entrée de l'état cible seront exécutés, mais pas ceux de l'état les contenant.

**Transition d'achèvement et événement d'achèvement (completion transition and completion event)**

• Une **transition d'achèvement** a un déclencheur (trigger) implicite, c’est une transition particulière.

• L'événement qui valide ce déclencheur est appelé **événement d'achèvement**.

• Dans le cas d'états simples, un événement d'achèvement est généré lorsque les comportements d'entrée (entry) et durant l’état actif (do) associés ont terminé leur exécution. Si aucun de ces comportements n'est défini, l'événement d'achèvement est généré lors de l'entrée dans l'état.

• Pour les états composites ou de sous-machine, un événement d'achèvement est généré dans les circonstances suivantes :

* toutes les activités internes (par exemple, les comportements d'entrée et durant un état actif) ont terminé leur exécution, et
* si l'état est un état composite, toutes ses régions orthogonales ont atteint un état final, ou
* si l'état est un état sous-machine, l'exécution de la machine à états de sous-machine a atteint un état final.

• Les **événements d'achèvement** ont la **priorité de distribution**. Autrement dit, ils sont distribués avant toute occurrence d'événement en attente dans le pool d'événements.

• Si deux événements d'achèvement ou plus correspondant à plusieurs régions orthogonales se produisent simultanément (c'est-à-dire à la suite de la même occurrence d'événement), l'ordre dans lequel ces occurrences d'achèvement sont traitées n'est pas défini.

• L'achèvement de toutes les régions de niveau supérieur dans une machine à états correspond à un achèvement du comportement de la machine à états et aboutit à sa terminaison.

**Garde de transition (transition guard)**

• Une transition peut avoir une **contrainte de garde** associée.

• Les transitions dont la garde est évaluée à faux (false) sont inhibées (non validées).

• Les gardes sont évaluées avant que la transition composée qui les contient ne soit validée, à moins qu'elles ne se trouvent sur des transitions qui proviennent d'un pseudo-état de choix. Dans ce dernier cas, les gardes sont évaluées lorsque le point de choix est atteint.

• Une transition qui n'a pas de garde associée est traitée comme si elle avait une garde toujours vraie (true).

• Une transition d'achèvement peut également avoir une garde.

• Une contrainte de garde peut impliquer des **tests d'états** (par exemple, « in state1 » ou « not in state2 »). Les noms d'état peuvent être entièrement qualifiés par les états imbriqués et les régions qui les contiennent, ce qui donne des chemins d'accès de la forme « regionA :: state1 :: region1 :: state2 :: state3 ».

**Transitions composées (compound transitions)**

• Lorsqu'une occurrence d'événement déclenche une transition validée ou qu'une exécution de machine d'état est créée, cela peut initier le parcours d'un ensemble de transitions et de sommets connectés et imbriqués jusqu'à ce qu'une configuration d'état stable soit atteinte. Une trace de ce parcours est appelée **transition composée**.

• Le comportement global qui résulte de l'exécution d'une transition composée est un ensemble partiellement ordonné d'exécutions de comportements associés aux éléments traversés, déterminé par l'ordre dans lequel les éléments (sommets et transitions) sont rencontrés. Par exemple, si une transition entrant dans un état composé se termine sur un sous-état de cet état, alors le comportement de la transition serait exécuté avant l'exécution du comportement d'entrée de l'état composé, suivi du comportement d'entrée du sous-état.

• Si un pseudo-état de fourche est rencontré dans le parcours, alors les comportements des différentes branches sortantes sont, au moins conceptuellement, exécutés simultanément les uns avec les autres.

**Traitement des événements pour les machines d'état (event processing for state machines) – Paradigme de l'exécution jusqu’à la fin, ou jusqu’à l'achèvement, ou de A à Z (run-to-completion paradigm – RTC paradigm)**

• Principes du traitement :

* Dans les machines à états SysML-UML, les transitions sont déclenchées par des occurrences d’événements.
* Une machine à états possède un espace de stockage des événements appelé « pool » d’événements.
* Initialisation :
  + La machine à états exécute sa transition (composée dans le cas général) initiale.
  + Puis elle entre en attente dans sa configuration d'état stable initiale.
* Durant toute la vie de la machine :
  + Dès l’occurrence d’un événement, la machine stocke l’information dans le « pool » (espace de stockage) d’événements.
  + En attente dans une configuration d'état (actif) stable, la machine choisit l’événement à traiter dans le « pool ». Alors :
    - La machine supprime l’événement choisi du « pool » d’événements (s’il n’est pas différé).
    - La machine traite l’événement (son occurrence) choisi dans le « pool » d’événements conformément aux règles d’évolution, entrant ainsi dans une phase transitoire.
    - Puis la machine entre à nouveau en attente, soit dans une nouvelle configuration d'état (actif) stable, soit dans la configuration d'état (actif) stable dans laquelle elle était.

• La machine à états traite une seule occurrence d'événement à la fois. Autrement dit, une occurrence d’événement ne peut être traitée que si le traitement de la précédente occurrence est terminé.

• Ce choix a été fait pour éviter les complications résultant de conflits de concurrence.

• Ce principe est appelé le paradigme comportemental de l'exécution jusqu’à la fin, ou jusqu’à l'achèvement **(run-to-completion paradigm – RTC paradigm)**.

• En conséquence, la machine passe d’une configuration d’état (actif) stable à une autre, entre deux traitements d’occurrences d’événements.

• Les événements d'achèvement ont la priorité et seront envoyés avant toute autre occurrence d'événement en attente dans le pool d'événements.

• Il est possible que plusieurs transitions s'excluant mutuellement dans une région donnée soient validées pour le déclenchement par la même occurrence d'événement. Dans ces cas, une seule est sélectionnée et exécutée.

• En raison de la présence de régions orthogonales, il est possible que plusieurs transitions (dans différentes régions) puissent être déclenchées par la même occurrence d'événement.

* L'ordre dans lequel ces transitions sont exécutées n'est pas défini.
* Chaque région orthogonale dans la configuration d'état actif qui ne contient pas de régions orthogonales imbriquées (c'est-à-dire, une région «de niveau inférieur») peut déclencher au plus une transition à la suite de l'occurrence d'événement en cours.
* Lorsque toutes les régions orthogonales ont fini d'exécuter la transition, l'occurrence d'événement en cours est entièrement consommée et l'étape de l'exécution jusqu'à la fin est terminée.

**Transitions validées (enabled transitions)**

• Une transition est validée si et seulement si :

* Tous ses états source sont dans la configuration d'état actif.
* Au moins l'un des déclencheurs de la transition a un événement qui correspond au type d'événement de l'occurrence d'événement distribuée.
* S'il existe au moins un chemin complet depuis la configuration de l'état source vers la configuration de l'état cible ou vers un pseudo-état de choix dynamique dans lequel toutes les conditions de garde sont vraies.

• Comme plus d'une transition peut être validée par la même occurrence d'événement, la validation est une condition nécessaire mais non suffisante pour le déclenchement d'une transition.

**Transitions conflictuelles (conflicting transitions)**

• Plusieurs transitions peuvent être validées dans une machine à états. Si cela se produit, ces transitions peuvent être en conflit. Par exemple, considérons le cas de deux transitions provenant du même état, déclenchées par le même événement, mais avec des gardes différentes. Si cet événement se produit et que les deux conditions de garde sont vraies, alors au plus une des deux transitions peut être déclenchée dans une étape d'exécution jusqu'à la fin.

• Rappelons que seules les transitions qui se produisent dans des régions mutuellement orthogonales peuvent être déclenchées simultanément.

**Priorités de tir (firing priorities)**

• En cas de transitions conflictuelles, la sélection des transitions qui se déclencheront repose en partie sur une priorité implicite. Ces priorités résolvent certains conflits de transition, mais pas tous.

• Les priorités des transitions conflictuelles sont basées sur leur position relative dans la hiérarchie des états. Par définition, une transition provenant d'un sous-état a une priorité plus élevée qu'une transition en conflit provenant de l'un de ses états le contenant. La priorité d'une transition est définie en fonction de son état source.

• La priorité des transitions enchaînées dans une transition composée est basée sur la priorité de la transition avec l'état source le plus profondément imbriqué.

En général, si t1 est une transition dont l'état source est s1, et t2 a pour source s2, alors :

* Si s1 est un sous-état imbriqué directement ou indirectement de s2, alors t1 a une priorité plus élevée que t2.
* Si s1 et s2 ne sont pas dans la même configuration d'état, alors il n'y a pas de différence de priorité entre t1 et t2.

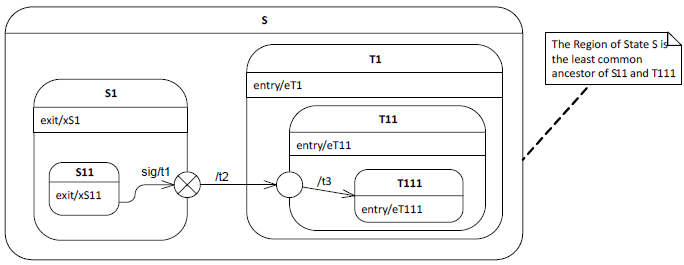
**Séquence d'exécution de la transition (transition execution sequence)**

• Chaque transition, à l'exception des transitions internes et locales, provoque la sortie d'un état source et l'entrée de l'état cible. Ces deux états, qui peuvent être composites, sont désignés respectivement comme source principale et cible principale d'une transition. La source principale est un sous-état direct de la région qui contient les états source, et la cible principale est le sous-état de la région qui contient les états cibles.

• Une fois qu'une transition est validée et sélectionnée pour se déclencher, les étapes suivantes sont exécutées dans l'ordre :

* 1) En commençant par l'état source principal, les états qui contiennent l'état source principal sont sortis selon les règles de sortie d'état (ou de sortie d'état composite si l'état source principal est imbriqué) comme décrit précédemment.
* 2) La série de sorties d'états se poursuit jusqu'à ce que la première région contenant, directement ou indirectement, à la fois les états source principal et cible principal soit atteinte. La région qui contient à la fois les états source principal et cible principal est appelée leur ancêtre le moins commun. À ce stade, le comportement de la transition qui relie la sous-configuration des états source à la sous-configuration des états cibles est exécuté.
* 3) La configuration des états contenant l'état cible principal est entrée, en commençant par l'état le plus extérieur dans la région ancêtre la moins commune qui contient l'état cible principal. L'exécution des comportements suit les règles d'entrée d'état (ou d'entrée d'état composite) décrites précédemment.

Cet algorithme d'exécution de transition est illustré par l'exemple suivant. Dans ce cas, lorsque l'événement « sig » est distribué alors que la machine est dans l'état « S11 » (la source principale), la séquence d'actions suivante sera exécutée: xS11; t1; xS1; t2; eT1; eT11; t3; eT111.



# Notes SysML

Une **note** permet d’apporter un commentaire sur un élément ou un lien quelconque du modèle.

|  |  |
| --- | --- |
|  | • « stéréotype » = « rationale » : indique une justification.  • « stéréotype » = « problem » : indique un problème non résolu.  • sans stéréotype : autre note. |

# Eléments sur les diagrammes des exigences

Le diagramme des exigences (requirement – req) spécifie ce qui est attendu du système. Diverses relations permettent de relier les exigences à d'autres exigences ainsi qu'à d'autres éléments du modèle.

|  |  |
| --- | --- |
|  | • Une exigence spécifie ce qui doit (ou devrait) être satisfait ; elle peut par exemple, spécifier une fonction qu'un système doit exécuter ou une condition de performance qu'un système doit atteindre.  • La partie texte décrit l’exigence (caractéristiques, qualités).  • Une priorité peut être associée à une exigence. |
|  | **Hiérarchisation d'exigences (contenance / inclusion) :** une exigence composite peut contenir des sous-exigences (hiérarchie). Cette relation permet à une exigence complexe d'être décomposée en ses exigences enfant.  • L’exigence A contient les exigences B et C ; les exigences B et C sont nécessaires à l’exigence A.  • Le cercle avec une croix est du côté de l’exigence de départ.  • Les exigences B et C peuvent être décomposées en sous-exigences, etc.  • Cela définit une hiérarchie des exigences. |
|  | **Copie d’exigence (Maitre-Esclave) :** pour réutiliser des exigences dans des applications différentes, SysML a introduit le concept d'exigence esclave. Une exigence esclave est une exigence dont la propriété text est une copie en lecture seule de la propriété text d'une exigence maître. La relation maître / esclave est indiquée par l'utilisation de la relation « copy ». |
|  | **Dérivation d’exigence :** cette relation relie une exigence dérivée à son exigence source.  • L’exigence dérivée (origine de la flèche) est reliée à son exigence source : B est dérivée de A ; B peut étendre A.  • Les exigences dérivées correspondent généralement aux exigences du niveau suivant de la hiérarchie du système.  • L’exigence dérivée peut l’être de plusieurs autres exigences.  Exemple : soit l’exigence d'accélération d’un véhicule. Les exigences dérivées peuvent être les exigences de puissance du moteur, de poids du véhicule, … |
|  | **Raffinement (ou précision) d’exigence :** cette relation peut être utilisée pour décrire comment un élément de modèle ou un ensemble d'éléments peut être utilisé pour affiner/préciser davantage une exigence. Il peut également être utilisé pour montrer comment une exigence basée sur du texte affine un élément de modèle.  • Le raffinement est relié à l’origine de la flèche : B raffine/précise A.  • L’exigence précisée est reliée à la pointe de la flèche ; elle peut l’être par plusieurs précisions. |
|  | **Satisfaction d’exigence :** cette relation décrit la manière dont un modèle de conception ou de mise en œuvre satisfait une ou plusieurs exigences. Elle permet de spécifier les éléments de conception du système destinés à satisfaire des exigences.  • L’élément B, reliée à l’origine de la flèche, satisfait l’exigence A. |
|  | **Vérification d’exigence :** cette relation de vérification définit la manière dont un scénario de test ou un autre élément de modèle vérifie une exigence.  • L’élément B permet de vérifier A. |
|  | **Trace d’exigence :** cette relation générique fournit une relation à usage général entre une exigence et tout autre élément du modèle.  • B est à l’origine de A. |

# Eléments sur les blocs, ports et flux

• Les **blocs** permettent de modéliser des systèmes sous forme d'arbres de composants modulaires. Ils peuvent inclure à la fois des caractéristiques structurelles et comportementales.

• Les **blocs** peuvent être utilisés à toutes les phases de la spécification et de la conception du système.

• La spécification des **ports** et des **flux** permet une conception de blocs modulaires, réutilisables, avec des moyens clairement définis de connexion et d'interaction avec leur contexte d'utilisation.

# Eléments sur les diagrammes de définition de bloc

Le diagramme de définition de bloc (block definition diagram – bdd) définit la décomposition hiérarchique du système, les blocs qui sont utilisés dans le diagramme de bloc interne, et leurs relations.

|  |  |
| --- | --- |
|  | **Bloc (block) :** c’est l’élément de base du diagramme.  • Il permet de représenter tout élément matériel, logiciel, abstrait.  • Il peut comporter plusieurs compartiments : contraintes (constraints), parties/composants (parts), propriétés (properties), références (references), valeurs (values), opérations (operations), comportements (behavior), …  • Alternativement, des compartiments peuvent aussi être représentés graphiquement avec des relations qui les lient au bloc. |
|  | **Type valeur (valueType) :** la définition de types de valeurs permet de spécifier les valeurs utilisables dans un modèle, par exemple : précision, échelle, virgule fixe ou flottante, entiers, ... |
| *Chemins graphiques définis dans les diagrammes de définition de bloc* | |
|  | **Association de parties/composants (part association) :** cette relation indique que le bloc de niveau supérieur (côté losange plein) est nécessairement composé des blocs de niveau inférieur.  • Le bloc A contient nécessairement les blocs B et C.  • Cette relation permet de représenter la décomposition d’un élément en sous-éléments.  • Cette relation peut être nommée et une multiplicité définie.  • Elle est aussi appelée relation de **contenance** ou de **composition** ou d’appartenance ou d’agrégation forte. |
|  | **Association partagée (shared association) :** cette relation indique que le bloc de niveau supérieur (côté losange vide) peut être composé des blocs de niveau inférieur mais pas obligatoirement.  • Le bloc A peut contenir le bloc B.  • Cette relation peut être nommée et une multiplicité définie.  • Elle est aussi appelée d’agrégation. |
|  | **Généralisation / spécialisation (generalization) :** cette relation indique que le bloc spécialisé (origine de la flèche) hérite de toutes les propriétés du bloc général (côté flèche) et peut éventuellement en posséder d’autres.  • Le bloc B hérite du bloc A. |
|  | **Association :** cette relation indique un lien entre deux blocs.  • Un lien bidirectionnel est représenté par un trait sans flèche.  • Un lien unidirectionnel est représenté par une flèche.  • Il n’y a pas de hiérarchie entre les blocs.  • Cette relation peut être nommée et une multiplicité définie. |

# Eléments sur les diagrammes de bloc interne

Le diagramme de bloc interne (internal block diagram – ibd) décrit la structure interne du système et les interactions entre les blocs du système. Les blocs sont définis dans le diagramme de définition de bloc.

|  |  |
| --- | --- |
|  | **Bloc (block) :** c’est l’élément de base du diagramme.  • Il est utilisé dans l’ibd et défini dans le bdd.  • Le bloc est désigné par nom : type où type est le nom du bloc défini dans le bdd. |
| *Chemins graphiques définis dans les diagrammes de bloc interne* | |
|  | **Connecteur (connector) :** un connecteur relie deux blocs.  • Il peut être nommé.  • Un connecteur bidirectionnel est représenté par un trait continu.  • Un connecteur unidirectionnel est représenté par une flèche.  • Il n’y a pas de hiérarchie entre les blocs.  • Une multiplicité peut être définie. |
|  | **Connecteur de liaison (binding connector) :** ce connecteur spécifie que les propriétés à ses deux extrémités ont des valeurs égales (de manière récursive via toutes les propriétés imbriquées le cas échéant). |

# Eléments sur les ports et flux

Les mêmes notations sont utilisées pour le bdd et l’ibd.

• SysML identifie deux modèles d'utilisation pour les ports : les ports « proxy » et les ports « complets ». Les deux sont des moyens de définir la limite du bloc propriétaire en tant que fonctionnalités disponibles via des connecteurs externes vers les ports. Dans les deux cas, les utilisateurs d'un bloc ne sont concernés que par les fonctionnalités de ses ports, que les fonctionnalités soient affichées par des ports proxy ou gérées directement par des ports complets.

• Les ports qui ne sont ni proxy ni complet sont simplement appelés « ports ».

*Note : les ports de flux et les spécifications de flux sont inclus dans SysML pour la compatibilité, mais sont obsolètes depuis la version 1.3. Ils ne sont pas décrits ici.*

|  |  |
| --- | --- |
|  | **Port :** un port est un point de connexion auquel des entités externes peuvent se connecter et interagir avec le bloc.  • Les fonctionnalités disponibles pour les entités externes via des connecteurs aux ports sont spécifiées par des propriétés.  • Les fonctionnalités peuvent être des propriétés de flux, des fins d'association, des opérations, des réceptions, ... |
|  | **Ports avec propriétés de flux (flow properties) :** les propriétés de flux spécifient les types d'éléments qui peuvent circuler entre un bloc et son environnement, qu'il s'agisse de données, de matériaux ou d'énergie. |
|  | **Port avec compartiments :** |
|  | **Port imbriqué (nested port) :**  • Les ports imbriquent les autres ports de la même manière que les blocs imbriquent d'autres blocs.  • Le type de port est un bloc qui possède également des ports. |
|  | **Ports proxy (proxy port) :** les ports proxy agissent comme des proxys pour leurs propres blocs ou ses composants internes.  *(Un proxy joue le rôle d'intermédiaire en se plaçant entre deux hôtes pour faciliter ou surveiller leurs échanges.)*  • Les ports proxy définissent la frontière en spécifiant quelles caractéristiques du bloc propriétaire ou des composants internes sont visibles via des connecteurs externes.  • Les ports proxy sont toujours typés par des blocs d'interface (interface block), un type de bloc spécialisé qui n'a aucun comportement ni aucun composant interne. |
|  | **Port complet (full port) :** les ports complets définissent la frontière avec leurs propres caractéristiques. |
|  | **Flux d'éléments (item flow) :** les flux d'éléments spécifient ce qui « circule » entre les blocs et / ou les composants dans un contexte d'utilisation particulier, alors que les propriétés de flux spécifient ce qui « entre » ou « sort » d'un bloc.  • Un flux d'éléments est représenté par une pointe de flèche noire sur le connecteur ou l'association. La pointe de la flèche est dirigée vers la cible.  • Pour un flux d'éléments avec une propriété d'élément, l'étiquette affiche le nom et le type de la propriété d'élément (nom: type). Sinon, le flux est étiqueté avec le nom du classificateur des éléments.  • Lorsque plusieurs flux d'éléments ayant la même direction sont représentés, un seul triangle est affiché et la liste des flux d'éléments, séparés par une virgule, est présentée. |
|  | **Interfaces requises et fournies (required and provided interfaces) :**  • Une interface requise sur un port spécifie une ou plusieurs fonctionnalités requises par les comportements du bloc (ou de ses composants).  • Une interface fournie sur un port spécifie une ou plusieurs fonctionnalités qu'un bloc (ou une ou plusieurs de ses composants) doit fournir.  • Un composant qui a un port avec une interface requise doit être connecté à un autre composant qui fournit les services dont il a besoin, généralement via un port avec une interface fournie.  • Ces interfaces sont « redondantes » avec les capacités des blocs d'interface mais elles sont conservées en SysML car elles sont utilisées en UML. |