ENTERPRISE ARCHITECT

Série de Guides d'Utilisateur

Statemachines Exécutables

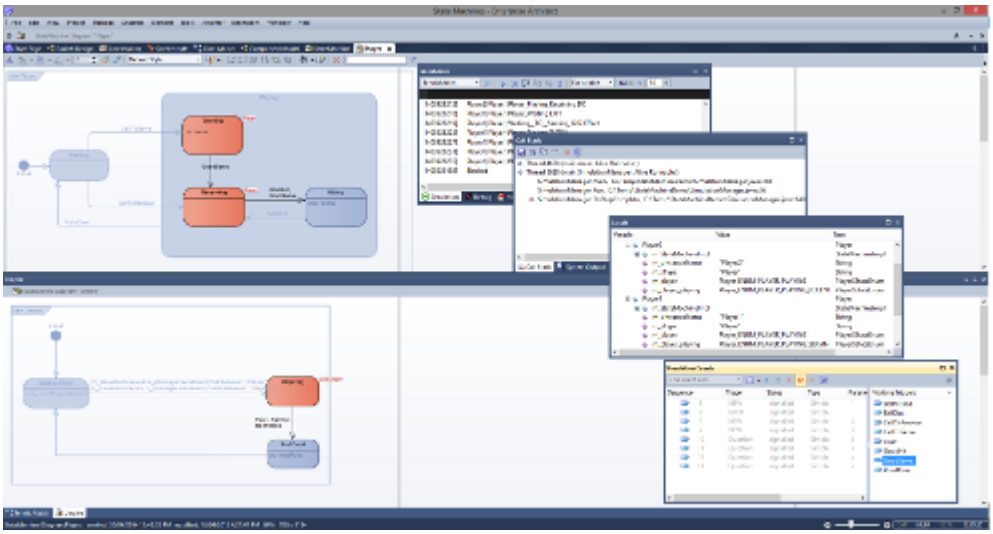
Author: Sparx Systems
Date: 23/11/2023
Version: 16.1

CRÉÉ AVEC ENTERPRISE ARCHITECT

Table des Matières

Statemachines Exécutables 3
- Macro d'événement : EVENT_PARAMETER 5
- Exemple : Simulation en HTML avec JavaScript 8
  - Lecteur CD 9
  - Parser d'expressions régulières 13
- Exemple : Commandes de Simulation 15
- Exemple : Entrer d'un State 23
- Exemple : Fourche et Joindre 32
- Exemple : Pseudo-state Historique 36
- Exemple : points d'entrée et de sortie (références de points de connexion) 44
- Exemple : Motif d'événement différé 49
- Débogage de l'exécution des Statemachines Exécutables 55
- Exemple : Statemachine Exécutable 57
- Artefact Statemachine Exécutable 61
- Exécution et Simulation de Statemachines Exécutables 63
- Génération de code pour Statemachines Exécutables 64
- Modélisation Statemachines Exécutables 70

Statemachines Exécutables



Statemachines Exécutables fournissent un moyen de générer, d'exécuter et de simuler rapidement des modèles d'état complexes. Contrairement à la simulation dynamique des graphiques State utilisant le moteur Simulation d' Enterprise Architect , Statemachines Exécutables fournissent une implémentation complète spécifique au langage qui peut former le « moteur » comportemental pour plusieurs produits logiciels sur plusieurs plates-formes. La visualisation de l'exécution est basée sur une intégration transparente avec la capacité Simulation . L'évolution du modèle présente maintenant moins de défis de codage. La génération, la compilation et l'exécution du code sont prises en charge par Enterprise Architect . Pour ceux qui ont des besoins particuliers, chaque langage est fourni avec un ensemble de gabarits de code. Gabarits peut être personnalisé par vous pour adapter le code généré comme bon vous semble.

Ces rubriques vous présentent les bases de modélisation Statemachines Exécutables et vous aident à comprendre comment les générer et les simuler.

La création et l'utilisation de Statemachines Exécutables , ainsi que la génération de code à partir de celles-ci, sont prises en charge par les éditions Unified et Ultimate d' Enterprise Architect .

Présentation de la création et de l'exécution Statemachines

Construire et utiliser Statemachines Exécutables est assez simple, mais nécessite un peu de planification et une certaine connaissance de la façon de relier les différents composants pour construire un modèle d'exécution efficace. Heureusement, vous n'avez pas besoin de passer des heures à définir le bon modèle et à corriger les erreurs de compilation avant de pouvoir commencer à visualiser votre conception.

Après avoir esquissé les grandes mécaniques de votre modèle, vous pouvez générer le code pour le piloter, le compiler, l'exécuter et le visualiser en quelques minutes. Ces points résument ce qui est requis pour commencer à exécuter et simuler Statemachines .

Facilité	Description
Construire des modèles de classe et State	La première tâche consiste à créer les modèles de classe et State UML standard qui décrivent les entités et le comportement à construire. Chaque classe d'intérêt dans votre modèle doit avoir sa propre Statemachine qui décrit les différents états et transitions qui régissent son comportement global.
Créer un artefact Statemachine Exécutable	Une fois que vous avez modélisé vos modèles de classes et State , il est temps de concevoir l'artefact Statemachine Exécutable . Cela décrira les classes et les objets impliqués, ainsi que leurs propriétés et relations initiales. Il s'agit du script de liaison qui relie plusieurs objets entre eux et détermine comment ceux-ci communiqueront au moment de l'exécution. Note qu'il est possible d'avoir deux objets ou plus dans un artefact Statemachine Exécutable en tant qu'instances d'une

	seule classe. Ceux-ci auront leur propre état et comportement au moment de l'exécution et pourront interagir si nécessaire.
Générer du code et compiler	Que vous utilisiez JavaScript , C++, Java ou C# , les capacités d'ingénierie d' Enterprise Architect vous fournissent un outil efficace, vous permettant de régénérer l'exécutable à tout moment, et sans perte du code personnalisé que vous auriez pu créer. C'est un avantage majeur sur la durée de vie d'un projet. Il convient probablement également de noter que l'intégralité de la base de code générée est indépendante et portable. En aucun cas le code n'est couplé à une quelconque infrastructure utilisée par le moteur de simulation.
Exécuter Statemachines	Alors, comment pouvons-nous voir comment ces Statemachines se comportent ? Une méthode consiste à construire la base de code pour chaque plate-forme, à l'intégrer dans un ou plusieurs systèmes, en examinant les comportements, « in situ », dans peut-être plusieurs scénarios de déploiement. Ou nous pouvons l'exécuter avec Enterprise Architect . Qu'il s'agisse de Java, JavaScript , C, C++ ou C# , Enterprise Architect se chargera de créer le runtime, l'hébergement de votre modèle, l'exécution de ses comportements et le rendu de toutes Statemachines .
Visualiser Statemachines	La visualisation Statemachine Exécutable s'intègre aux outils Simulation d' Enterprise Architect . Observez les transitions d'état au fur et à mesure qu'elles se produisent sur votre diagramme et pour quel(s) object (s). Identifiez facilement les objets partageant le même état. Il est important de noter que ces comportements restent cohérents sur plusieurs plateformes. Vous pouvez également contrôler la vitesse à laquelle les machines fonctionnent pour mieux comprendre la chronologie des événements.
Déboguer Statemachines	Lorsque les états devraient changer mais ne le font pas, lorsqu'une transition ne devrait pas être activée mais l'est, lorsque le comportement est - en bref - indésirable et n'est pas immédiatement apparent du modèle, nous pouvons nous tourner vers le débogage. Analyseur d'Exécution Visuelle d' Enterprise Architect est livré avec des débogueurs pour tous les langages pris en charge par la génération de code ExecutableStateMachine. Le débogage offre de nombreux avantages, dont l'un pourrait être de vérifier/corroborer le code attaché aux comportements dans une Statemachine pour s'assurer qu'il est réellement reflété dans le processus d'exécution.

Macro d'événement : EVENT_PARAMETER

EVENT_PARAMETER est une macro de fonction utilisée pour accéder à l'attribut d'une instance de signal, dans le comportement de State , la garde et l'effet de Transition. Cette macro sera étendue en code exécutable selon le langage de simulation.

Accéder à la définition de macro par défaut

Ruban | Développer | Code source | Options | Modifier le code Gabarits | Langue | Paramètre d'événement Stm

Format d'utilisation

%EVENT_PARAMETER(SignalType, SignalAttributeName)%

Par exemple : Un signal 'MySignal' a deux attributs, 'foo: int ' et 'bar: int ' ; ces cas d'utilisation sont valides :

- Effet de la transition : %EVENT_PARAMETER(MySignal, foo)%
- Comportement de State : %EVENT_PARAMETER(MySignal, bar)%
- Garde de transition : %EVENT_PARAMETER(MySignal, bar)% > 10
- Comportement de State : %EVENT_PARAMETER(MySignal, bar)%++
- Tracez la valeur dans la fenêtre Simulation : %TRACE(EVENT_PARAMETER(MySignal, foo))%

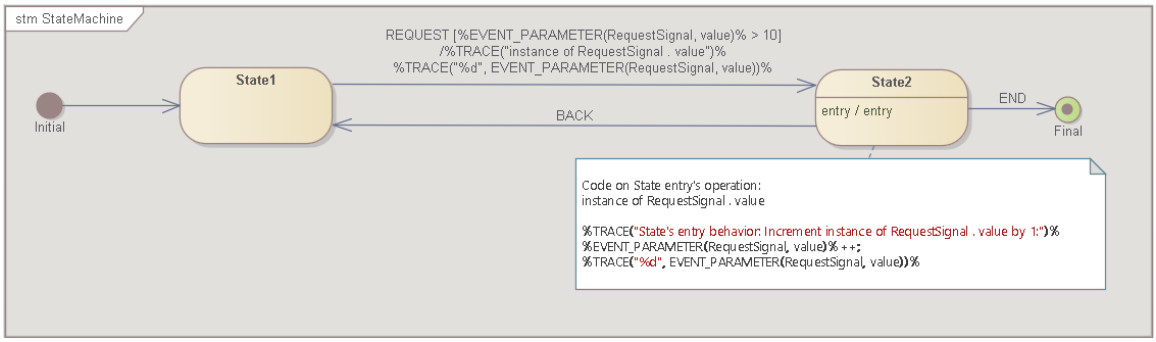
Exemple d'extension de macro

Pour le Signal "MySignal" avec attribut " valeur ", Exemples d'expansion de macro pour Transition avec déclencheur du signal :%EVENT_PARAMETER(MySignal, valeur)%

C	((MySignal*)signal) -> valeur
C++	static_cast<MySignal*>(signal)-> valeur
C#	((MonSignal)signal). valeur
Java	((EventProxy.MySignal)signal). valeur
JavaScript	signal. valeur

Exemple

Cet exemple a montré comment utiliser EVENT_MACRO dans l'effet, la garde et le comportement de State .



Lors de l'exécution de la simulation,

(1) déclencheur REQUEST et préciser le numéro 1 pour l'attribut valeur .

Puisque la condition pour la garde est fausse, l'état actif restera comme State1.

(2) déclencheur REQUEST et préciser le numéro 11 pour l'attribut valeur .

Puisque la condition de la garde est vraie, l'état actif passera de l'état 1 à l'état 2 ;

L'effet de la transition est exécuté. Ici, nous avons tracé la valeur d'exécution de l'attribut du signal jusqu'à la fenêtre de simulation.

Le comportement de State2 est exécuté. Ici, nous avons incrémenté la valeur d'exécution pour l'attribut du signal et l'avons tracé jusqu'à la fenêtre de simulation.

[32608107] [Part1:TransactionServer] Effet de transition : Initial_4019__TO__State1_4420

[32608118] Comportement d'entrée [Part1:TransactionServer] : StateMachine_State1

[32608124] [Part1:TransactionServer] Comportement d'exécution : StateMachine_State1

[32608877] [Partie 1 : TransactionServer] Achèvement : TransactionServer_StateMachine_State1

[32608907] En attente du Déclencheur

[32613165] Commande : **diffusion REQUEST.RequestSignal(1)**

[32613214] [Part1:TransactionServer] Événement mis en file d'attente : REQUEST.RequestSignal(valeur :1)

[32613242] [Part1:TransactionServer] Événement distribué : REQUEST.RequestSignal(valeur :1)

[32613279] En attente du Déclencheur

[32619541] Commande : **diffusion REQUEST.RequestSignal(11)**

[32619546] [Part1:TransactionServer] Événement mis en file d'attente : REQUEST.RequestSignal(valeur :11)

[32619551] [Part1:TransactionServer] Événement distribué : REQUEST.RequestSignal(valeur :11)

[32619557] [Part1:TransactionServer] Comportement de sortie : StateMachine_State1

[32619562] [Part1:TransactionServer] Effet de transition : State1__TO__State2_4421

[32619567] **instance de RequestSignal . valeur**

[32619571] **11**

[32619576] Comportement d'entrée [Part1:TransactionServer] : StateMachine_State2

[32619584] **Comportement d'entrée de State : incrémentation de l'instance de RequestSignal . valeur par 1 :**

[32619590] **12**

[32619594] [Part1:TransactionServer] Comportement d'exécution : StateMachine_State2

[32620168] [Partie 1 : TransactionServer] Achèvement : TransactionServer_StateMachine_State2

[32620211] En attente du Déclencheur

[32622266] Commande : diffusion FIN

[32622272] [Part1:TransactionServer] Événement mis en file d'attente : FIN

[32622310] [Part1:TransactionServer] Événement distribué : FIN

[32622349] [Part1:TransactionServer] Comportement de sortie : StateMachine_State2

[32622359] [Part1:TransactionServer] Effet de transition : State2__TO__Final_4023_4423

[32622896] [Partie 1 : TransactionServer] Achèvement : TransactionServer_VIRTUAL_SUBMACHINESTATE

Limite et solution de contournement

Étant donné que l'expansion de la macro implique une conversion de type, il est de la responsabilité de l'utilisateur de s'assurer que la conversion de type est valide.

Et voici des solutions de contournement pour certains cas courants dans lesquels la conversion de type rencontrera des problèmes dans le modèle.

- **Une transition a plusieurs déclencheurs de différents types de signaux**

Par exemple, une transition a triggerA (specified SignalA) et triggerB (specified SignalB) ; Une macro %EVENT_PARAMETER(SignalA,attributOfA)% ne fonctionnera pas lorsque cette transition est déclenchée par triggerB.

Nous suggérons de créer deux transitions, une avec triggerA (SignalA spécifié) et l'autre avec triggerB (SignalB).

- **Un State est la cible de multiples transitions de différents déclencheurs de signaux**

Par exemple, TransitionA et TransitionB ciblent tous deux MyState, TransitionA est déclenchée par SignalA et TransitionB est déclenchée par SignalB.

Si EVENT_PARAMETER est utilisé dans le code de comportement de l'état, une macro ne pourra pas fonctionner dans les deux cas.

Nous suggérons de déplacer la logique traitant de l'attribut du signal du comportement de State vers l'effet de Transition.

- **Personnaliser gabarit et le code généré**

L'utilisateur peut également modifier le gabarit par défaut pour ajouter Exécuter Time Type Identification (RTTI) avant la conversion de type. L'utilisateur peut également modifier le code généré après la génération, ce qui peut également satisfaire l'objectif de simulation après compilation.

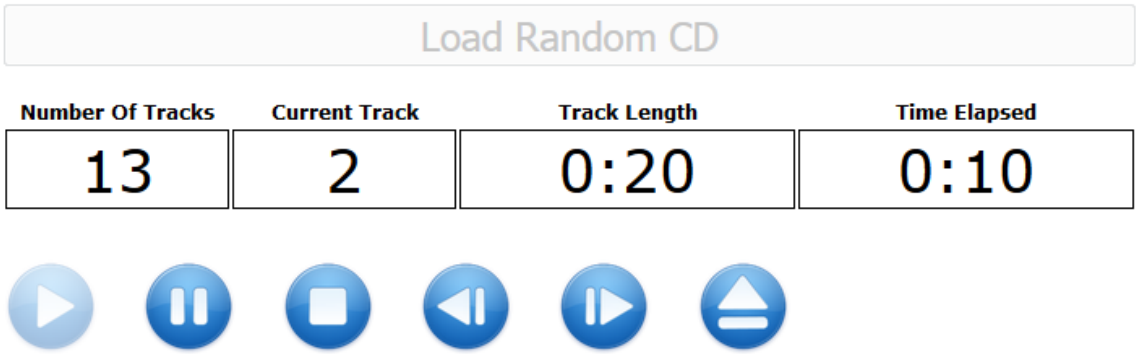
Exemple : Simulation en HTML avec JavaScript

Nous savons déjà que les utilisateurs peuvent modéliser un Statemachine Exécutable et le simuler dans Enterprise Architect avec le code généré. En utilisant les deux exemples *de lecteur CD* et *Parser d'expressions régulières* , nous allons maintenant démontrer comment vous pouvez intégrer le code généré à vos projets réels.

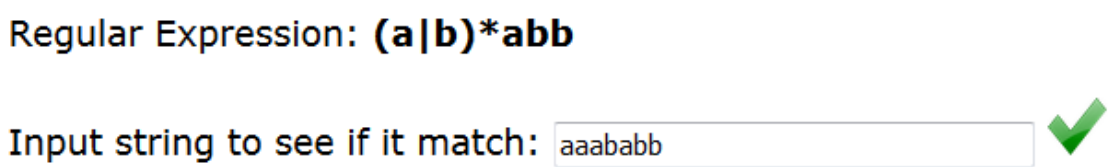
Enterprise Architect fournit deux mécanismes différents permettant au code client d'utiliser une Statemachine :

- Actif State Based - le client peut interroger l'état actif actuel, puis « changer » la logique en fonction du résultat de la requête.
- Basé sur une variable d'exécution : le client n'agit pas sur l'état actif actuel, mais agit sur la valeur d'exécution des variables définies dans la classe contenant la Statemachine .

Dans l'exemple *du lecteur CD* , il y a très peu d'états et de nombreux boutons sur l'interface graphique, il est donc assez facile d'implémenter l'exemple basé sur le mécanisme State Actif ; nous interrogerons également la valeur d'exécution pour la piste actuelle.



Dans l'exemple *Parser d'expressions régulières* , Statemachine gère tout et une variable membre *bMatch* change sa valeur d'exécution lorsque les états changent. Le client n'enregistre pas le nombre d'états présents ni l'état actuellement actif.



Dans ces rubriques, nous montrons comment modéliser, simuler et intégrer un lecteur de CD et un Parser pour une expression régulière spécifiée, étape par étape :

- CD Player
- Regular Expression Parser

Lecteur CD

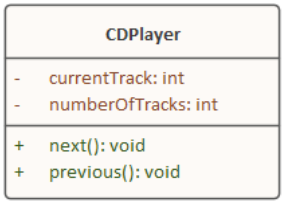
Le comportement d'une application de lecteur de CD peut sembler intuitif ; cependant, il existe de nombreuses règles relatives au moment où les boutons sont activés et désactivés, à ce qui est affiché dans les champs de texte de la fenêtre et à ce qui se passe lorsque vous fournissez des événements à l'application.

Supposons que notre exemple de lecteur CD possède ces fonctionnalités :

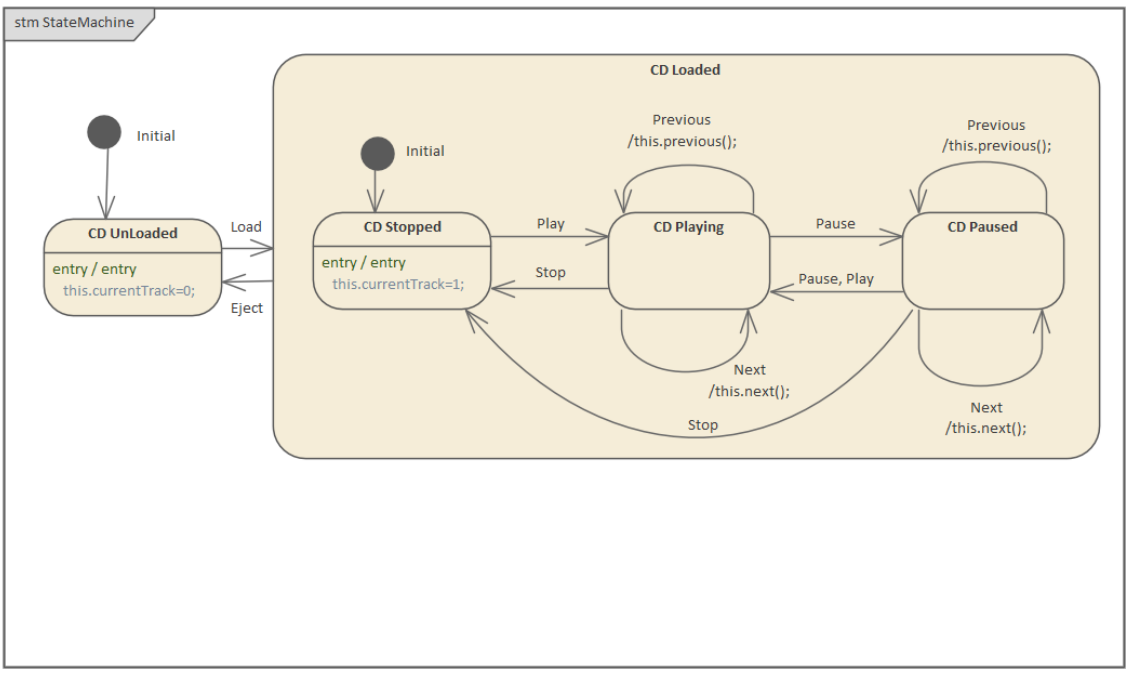
- Boutons - Charger un CD aléatoire, Lecture, Pause, Arrêt, Piste précédente, Piste suivante et Éjecter
- Affichages - Nombre de pistes, piste actuelle, longueur de piste et temps écoulé

Statemachine pour lecteur CD

Une classe *CDPlayer* est définie avec deux attributs : *currentTrack* et *numberOfTracks* .



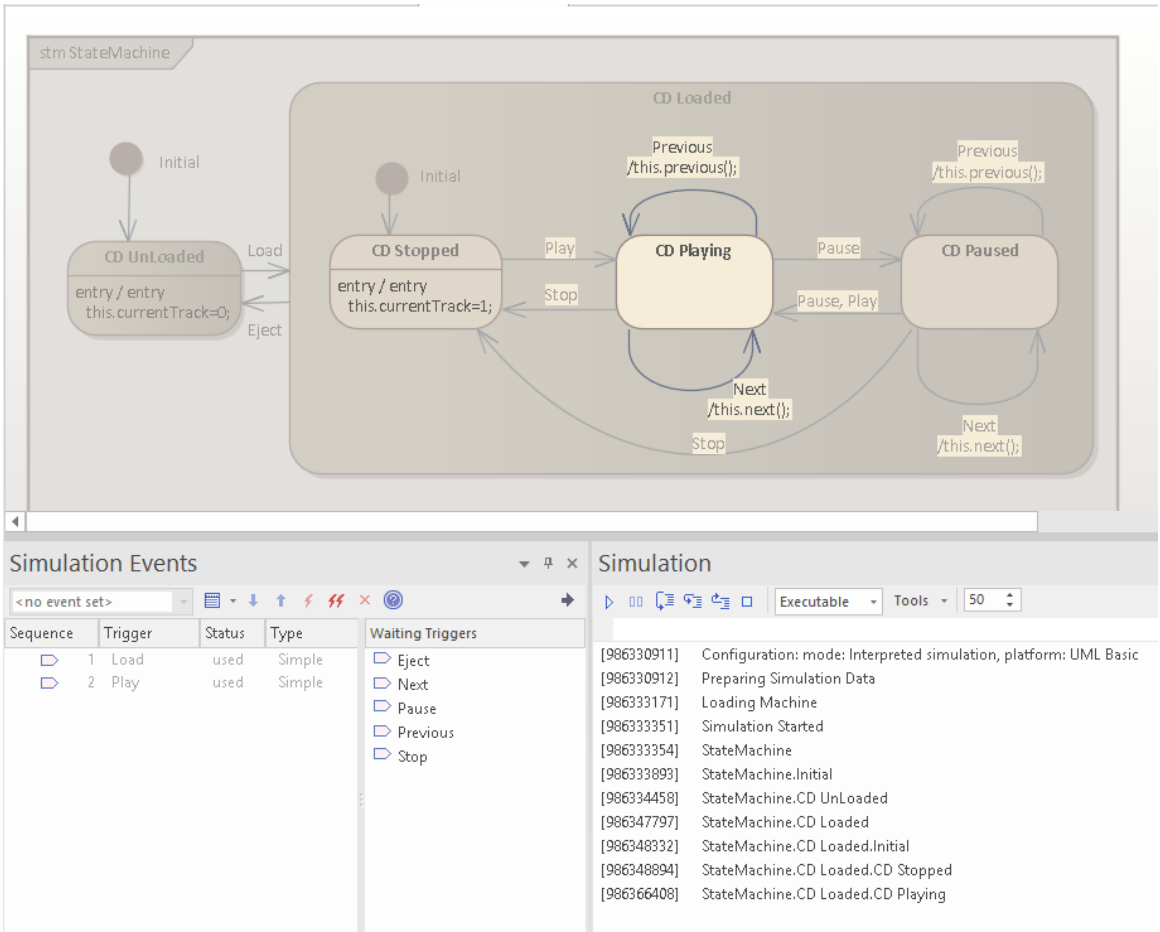
Une Statemachine est utilisée pour décrire les états du lecteur CD :



- Au niveau supérieur, la Statemachine possède deux States : *CD UnLoaded* et *CD Loaded*
- *Le CD chargé* peut être composé de trois States simples : *CD arrêté* , *lecture de CD* , *CD en pause.*
- Les transitions sont définies avec déclencheurs pour les événements Load, Eject, Play, Pause, Stop, Previous et Next
- Comportements State et effets de transition sont définis pour changer la valeur des attributs définis dans *CDPlayer* ; par exemple, l'événement 'Previous' déclencheur l'auto transition (si l'état actuel est *CD Playing* ou *CD Paused*) et l'effet sera exécuté, ce qui décrémentera la valeur de *currentTrack* ou reviendra à la dernière piste.

Nous pouvons créer un artefact Statemachine Exécutable et créer une propriété en tapant *CDPlayer* , puis simuler la

Statemachine dans Enterprise Architect pour nous assurer que le modèle est correct.



Inspecter le code généré

Enterprise Architect générera ces fichiers dans un dossier que vous avez spécifié :

- Code back-end : CDPlayer.js, ContextManager.js, EventProxy.js
- Code client : ManagerWorker
- Code frontal : statemachineGUI.js, index.html
- Autre code : SimulationManager.js

Déposer	Description
/CDPlayer.js	Ce fichier définit la classe CDPlayer et ses attributs et opérations. Il définit également les Statemachines de la classe avec les comportements State et les effets de transition.
/ContextManager.js	Ce fichier est le gestionnaire abstrait des contextes. Le fichier définit le contenu indépendant des contextes réels, qui sont définis dans la généralisation de *ContextManager* , tels que *SimulationManager* et *ManagerWorker* . La simulation (Statemachine Exécutable Artifact) peut impliquer plusieurs contextes ; par exemple, dans une simulation de jeu de tennis, il y aura un *arbitre* saisi en tant *qu'arbitre* de classe et deux joueurs - *joueurA* et *joueurB* - saisis en *joueur de classe. L'arbitre* de classe et *le joueur* de classe définiront leur(s) propre(s) machine(s) Statemachine .

/EventProxy.js	Ce fichier définit Événements et les Signaux utilisés dans la simulation. Si nous déclenchons un événement avec des arguments, nous modélisons l'événement comme un événement signal, qui spécifie une classe de signal ; nous définissons ensuite les attributs de la classe de signal. Chaque occurrence d'événement possède une instance du signal, transportant les valeurs d'exécution spécifiées pour les attributs.
/SimulationManager.js	Ce fichier est destiné à la simulation dans Enterprise Architect .
/html/ManagerWorker.js	Ce fichier sert de couche intermédiaire entre le front-end et le back-end. • Le front-end publie un message pour demander des informations au ManagerWorker • Étant donné que ManagerWorker généralise à partir de ContextManager, il a un accès complet à tous les contextes tels que l'interrogation de l'état actif actuel et l'interrogation de la valeur d'exécution d'une variable. • Le ManagerWorker publiera un message sur le front-end avec les données qu'il a récupérées du back-end
/html/statemachineGUI.js	Ce fichier établit la communication entre le front-end et le ManagerWorker, en définissant *stateMachineWorker* . Il: • Définit les fonctions *startStateMachineWebWorker* et *stopStateMachineWebWorker* • Définit les fonctions *onActiveStateResponse* et *onRuntimeValueResponse* avec un code d'espace réservé : //à faire : écrire la logique de l'utilisateur Vous pouvez simplement remplacer ce commentaire par votre logique, comme cela sera démontré plus loin dans cette rubrique.
/html/index.html	Celui-ci définit l' Interface Utilisateur , comme les boutons et les entrées permettant de déclencher Événements ou d'afficher des informations. Vous pouvez définir CSS et JavaScript dans ce fichier.

Personnaliser index.html et statemachineGUI.js

Apportez ces modifications aux fichiers générés :

- Créer des boutons et des affichages
- Créez un style CSS pour formater l'affichage et activer/désactiver les images des boutons
- Créez un ElapseTimeWorker.js pour actualiser l'affichage toutes les secondes
- Créez une fonction TimeElapsed, définie sur Next Track lorsque le temps écoulé atteint la longueur de la piste
- Créez JavaScript en tant que gestionnaire d'événements du bouton 'onclick'
- Une fois qu'un événement est diffusé, demandez l' State actif et valeur d'exécution pour *cdPlayer.currentTrack*
- A l'initialisation, demander l' State actif

Dans statemachineGUI.js, recherchez la fonction *onActiveStateResponse_cdPlayer*

- Dans CDPlayer_StateMachine_CDUnLoaded, désactivez tous les boutons et activez btnLoad
- Dans CDPlayer_StateMachine_CDLoaded_CDStopped, désactivez tous les boutons et activez btnEject et btnPlay
- Dans CDPlayer_StateMachine_CDLoaded_CDPlaying, activez tous les boutons et désactivez btnLoad et btnPlay

- Dans CDPlayer_StateMachine_CDLoaded_CDPaused, activez tous les boutons et désactivez btnLoad

Dans statemachineGUI.js, recherchez la fonction *onRuntimeValueResponse*

- Dans *cdPlayer.currentTrack* , nous mettons à jour l'affichage de la piste actuelle et de la longueur de la piste

L'exemple complet

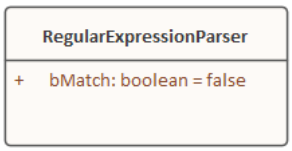
L'exemple est accessible depuis la page 'Ressources' du site Sparx Systems , en cliquant sur ce lien :
CD Player Simulation

Cliquez sur le bouton Charger un CD aléatoire, puis sur le bouton Démarrer Simulation .

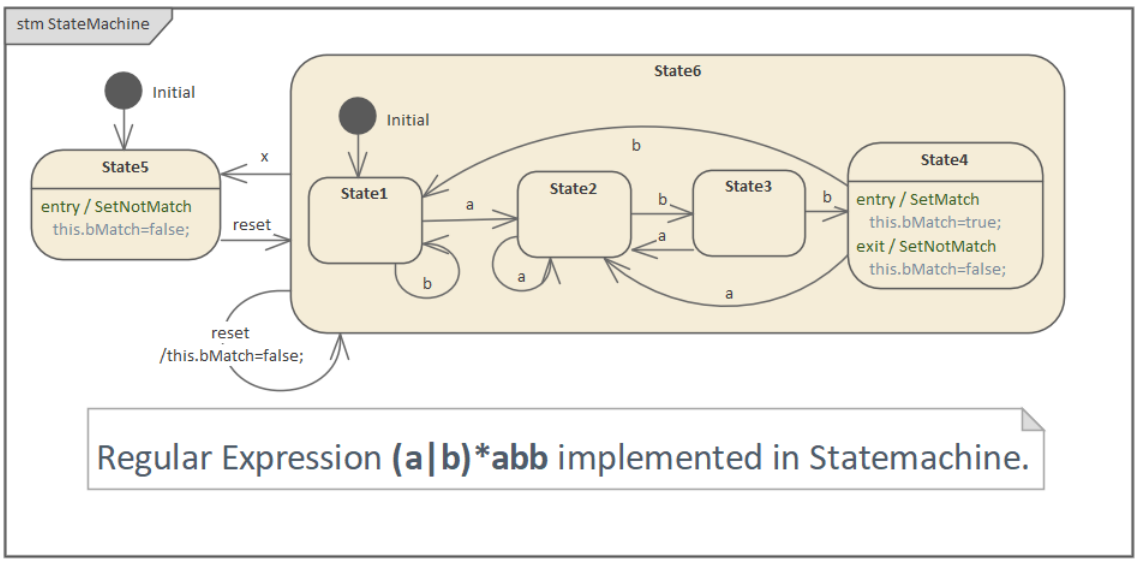
Parser d'expressions régulières

Statemachine pour Parser d'expressions régulières

La classe *RegularExpressionParser* est définie avec un attribut : bMatch.



Une Statemachine est utilisée pour décrire l'expression régulière (a|b)*abb



- Les déclencheurs de transition sont spécifiés comme événements *a* , *b* , *x* et *reset*
- À l'entrée dans State4, bMatch est défini sur True ; à la sortie de State4, bMatch est défini sur False
- À l'entrée dans State5, bMatch est défini sur False
- Lors de l'auto-transition de State6, bMatch est défini sur False

Personnaliser index.html et statemachineGUI.js

Apportez ces modifications aux fichiers générés :

- Créez un champ de saisie HTML et une image pour indiquer le résultat
- Créer JavaScript en tant que gestionnaire d'événements *oninput* du champ
- Créez la fonction 'SetResult ′ pour basculer l'image réussite/échec
- Créez la fonction 'getEventStr', qui renverra 'a' sur ' *a* ' et 'b' sur ' *b* ', mais renverra 'x' sur tout autre caractère
- A l'initialisation, diffuser ' *reset* '
- Lors de l'événement de diffusion, demandez la variable d'exécution 'regxParser.bMatch'

Dans *statemachineGUI.js* , recherchez la fonction 'onRuntimeValueResponse'.

- Dans 'regxParser.bMatch', nous recevrons 'True' ou 'False' et le transmettrons à 'SetResult' pour mettre à jour l'image

L'exemple complet

L'exemple est accessible depuis la page 'Ressources' du site Sparx Systems , en cliquant sur ce lien :
Regular Expression Parser Simulation

Exemple : Commandes de Simulation

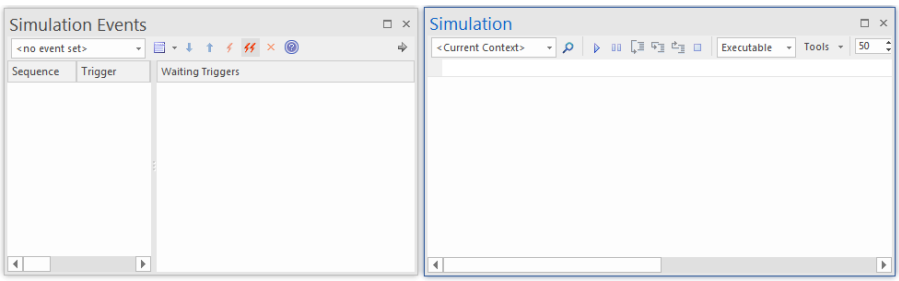
Cet exemple montre comment nous pouvons utiliser la fenêtre Simulation pour observer les messages Trace ou envoyer des commandes pour contrôler une Statemachine . Grâce à l'exemple, vous pouvez examiner :

- Un attribut d'un contexte - la variable membre définie dans la Class, qui est le contexte de la Statemachine ; ces attributs portent des valeurs dans le cadre du contexte pour tous les comportements State et effets de transition, pour accéder et modifier
- Chaque attribut d'un signal - la variable membre définie dans le signal, qui est référencée par un événement et peut servir de paramètre d'événement ; chaque occurrence d'événement de signal peut avoir différentes instances d'un signal
- L'utilisation de la commande 'Eval' pour interroger la valeur d'exécution d'un attribut de contexte
- L'utilisation de la commande « Dump » pour vider le décompte actif de l'état actuel ; il peut également vider l'événement en cours différé dans le pool

Cet exemple est tiré du modèle EAExample :

Exemple Modèle . Simulation de Modèle . Statemachine Exécutable . Commandes de Simulation

Accéder

Ruban	• Simuler > Simulation Dynamique > Simulateur > Ouvrir la fenêtre Simulation) • Simuler > Simulation Dynamique > Événements (pour la fenêtre Simulation Événements)  Ces deux fenêtres sont fréquemment utilisées ensemble dans la simulation de Statemachines Exécutables .

Créer un contexte et Statemachine

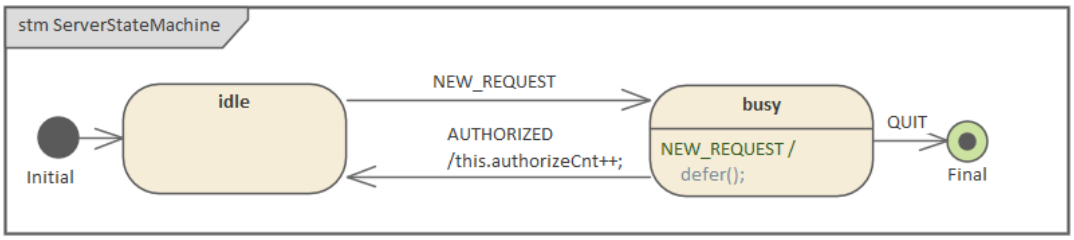
Dans cette section, nous allons créer une classe appelée TransactionServer, qui définit une Statemachine comme comportement. Nous créons ensuite un artefact Statemachine Exécutable comme environnement de simulation.

Créer le contexte de la Statemachine

1. Créez un élément Class appelé *TransactionServer.*

2. Dans cette classe, créez un attribut appelé *authorizeCnt* avec valeur initiale 0.
3. Dans la fenêtre Navigateur , cliquez-droit sur *TransactionServer* et sélectionnez l'option 'Ajouter | Option Statemachine .

Créer la Statemachine



1. Créez un pseudo-état initial appelé *Initial* .
2. Transition vers un State dit *de ralenti* .
3. Transition vers un State dit *occupé* , avec le déclencheur NEW_REQUEST.
4. Transition:
   - Vers un pseudo-état Final appelé *Final* , avec le déclencheur QUIT
   - Retour au *ralenti* , avec le déclencheur AUTHORIZED, avec l'effet 'this.authorizeCnt++;'

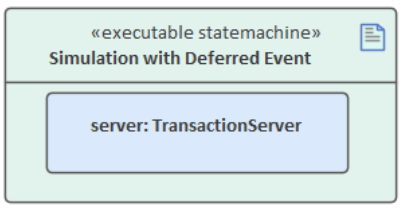
Créer un événement différé pour l' State *occupé*

1. Dessinez une auto-transition pour *être occupé.*
2. Changez le « genre » de la transition en « interne ».
3. Spécifiez le Déclencheur comme étant l'événement que vous souhaitez différer.
4. Dans le champ « Effet », tapez « defer(); ».

Créer un signal et Attributes

1. Créez un élément Signal appelé *RequestSignal.*
2. Créez un attribut appelé *requestType* avec le type ' int '.
3. Configurez l'événement NEW_REQUEST pour référencer *RequestSignal.*

Créer l'artefact Statemachine Exécutable

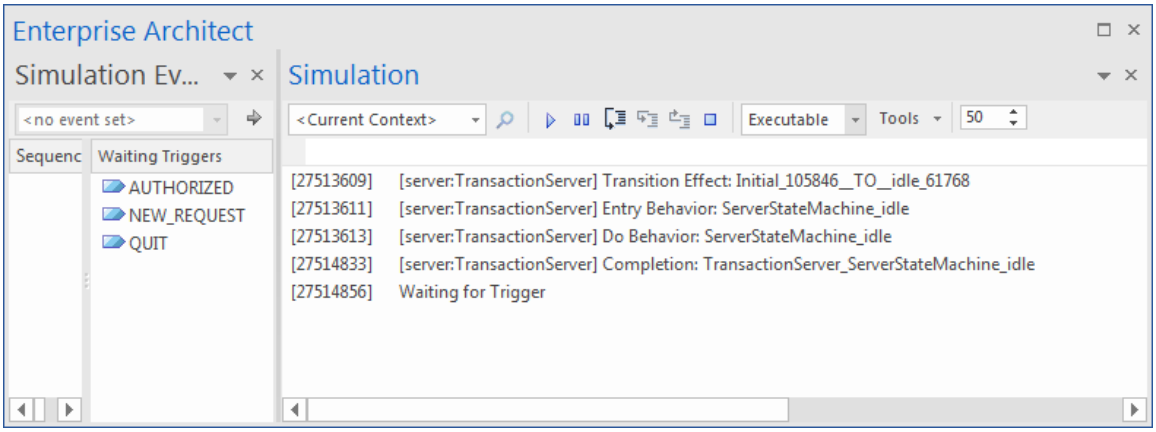


1. Depuis la page ' Simulation ' de la Diagramme Toolbox, faites glisser une icône ' Statemachine Exécutable ' sur le diagramme et appelez l'élément *Simulation with Deferred Event.*
2. Ctrl+Faites glisser l'élément *TransactionServer* depuis la fenêtre Navigateur et déposez-le sur l'Artefact en tant que propriété, avec le *serveur de noms.*
3. Définissez le langage de l'artefact sur JavaScript , qui ne nécessite pas de compilateur (par exemple ; en production, vous pouvez également utiliser C, C++, C# ou Java, qui prennent également support Statemachines Exécutables).
4. Cliquez sur l'artefact et sélectionnez l'option de ruban 'Simulate > États Exécutables > Statemachine > Générer , Build and Exécuter '.

Fenêtre Simulation et commandes

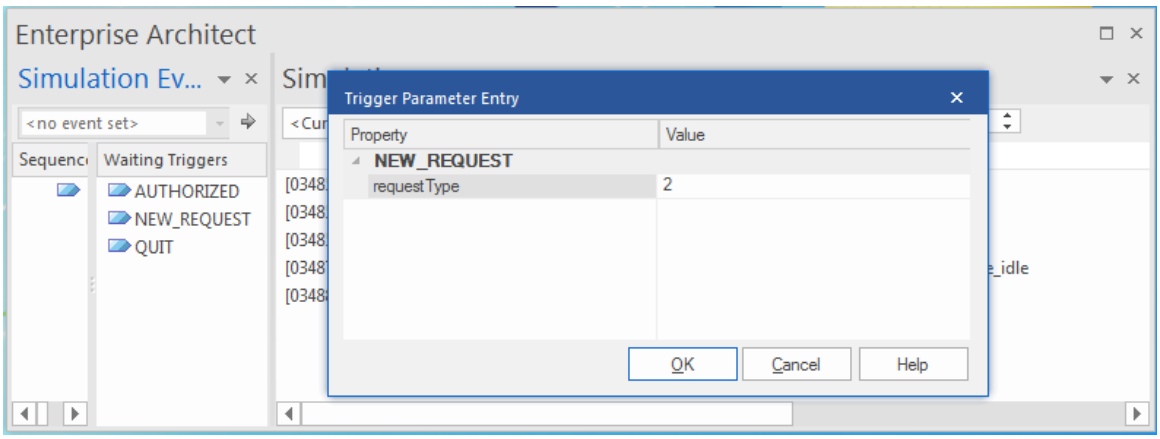
Lorsque la simulation démarre, *inactif* est l'état actuel.

La fenêtre Simulation montre que les comportements Transition Effect, Entry et Do sont terminés pour l'état *inactif* et que la Statemachine attend un déclencheur .



Données d'événement via des valeurs pour Attributes de signal

Pour l'événement de signal Déclencheur NEW_REQUEST, la dialogue « Entrée des paramètres Déclencheur » s'affiche pour prompt des valeurs pour les attributs répertoriés définis dans le Signal *RequestSignal* , référencés par NEW_REQUEST.



Type la valeur '2' et cliquez sur le bouton OK . Les valeurs de l'attribut Signal sont ensuite transmises aux méthodes invoquées telles que les comportements du State et les effets de la Transition.

Ces messages sont affichés dans la fenêtre Simulation :

[03612562] En attente du Déclencheur

[03611358] Commande : **diffusion NEW_REQUEST.RequestSignal(2)**

[03611362] [server:TransactionServer] Événement mis en file d'attente : NEW_REQUEST.RequestSignal(requestType:2)

[03611367] [server:TransactionServer] Événement distribué : NEW_REQUEST.RequestSignal(requestType:2)

[03611371] [server:TransactionServer] Comportement de sortie : ServerStateMachine_idle

[03611381] [server:TransactionServer] Effet de transition : ralenti__TO__busy_61772

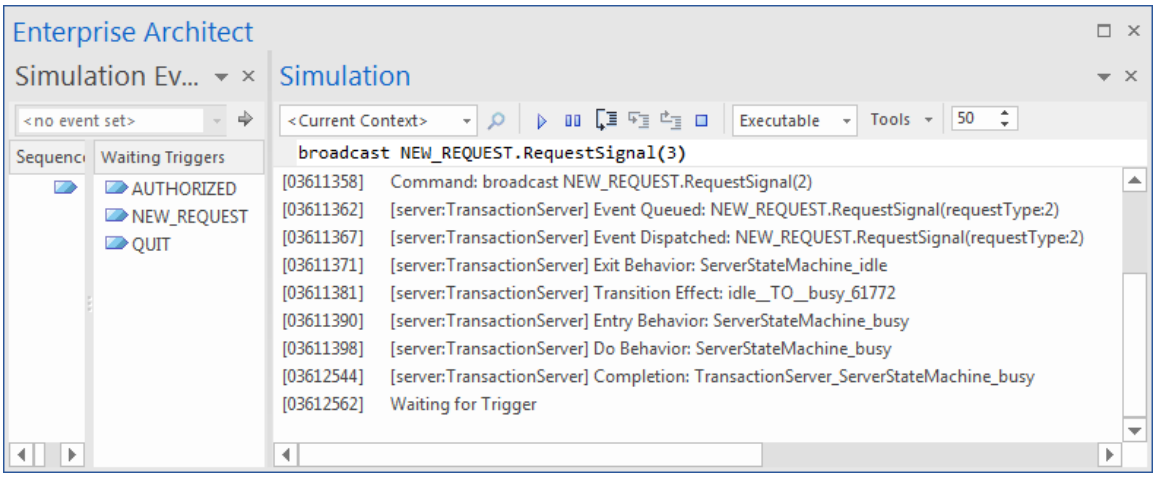
[03611390] [server:TransactionServer] Comportement d'entrée : ServerStateMachine_busy

[03611398] [server:TransactionServer] Comportement d'exécution : ServerStateMachine_busy

[03612544] [server:TransactionServer] Achèvement : TransactionServer_ServerStateMachine_busy

[03612562] En attente du Déclencheur

Nous pouvons diffuser des événements en double-cliquant sur l'élément répertorié dans la fenêtre Simulation Événements . Alternativement, nous pouvons taper une string de commande dans le champ de texte de la fenêtre Simulation (sous la barre d'outils).



[03612562] En attente du Déclencheur

[04460226] Commande : **diffusion NEW_REQUEST.RequestSignal(3)**

[04460233] [server:TransactionServer] Événement mis en file d'attente : NEW_REQUEST.RequestSignal(requestType:3)

[04461081] En attente du Déclencheur

Le message Simulation indique que l'occurrence de l'événement est différée (événement mis en file d'attente, mais non distribué). Nous pouvons exécuter d'autres commandes en utilisant le champ de texte :

[04655441] En attente du Déclencheur

[04664057] Commande : **diffusion NEW_REQUEST.RequestSignal(6)**

[04664066] [server:TransactionServer] Événement mis en file d'attente : NEW_REQUEST.RequestSignal(requestType:6)

[04664803] En attente du Déclencheur

[04669659] Commande : **diffusion NEW_REQUEST.RequestSignal(5)**

[04669667] [server:TransactionServer] Événement mis en file d'attente : NEW_REQUEST.RequestSignal(requestType:5)

[04670312] En attente du Déclencheur

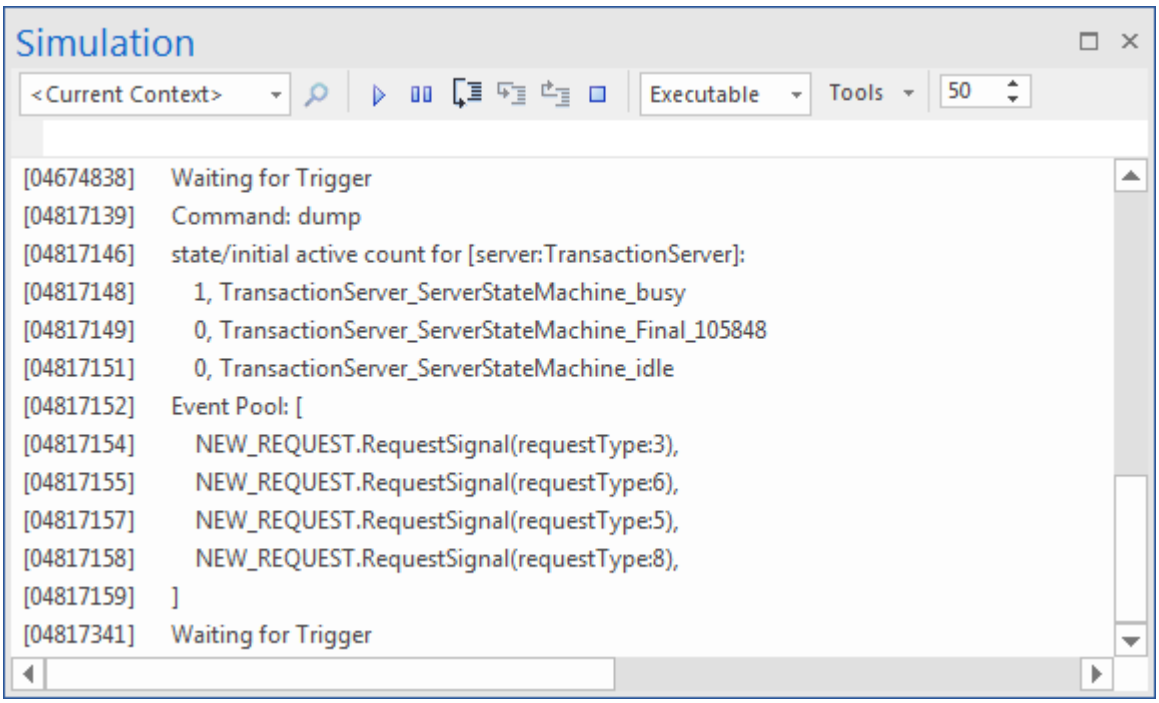
[04674196] Commande : **diffusion NEW_REQUEST.RequestSignal(8)**

[04674204] [server:TransactionServer] Événement mis en file d'attente : NEW_REQUEST.RequestSignal(requestType:8)

[04674838] En attente du Déclencheur

dump : Query « compte actif » pour un pool State et d'événements

Type *dump* dans le champ de texte ; ces résultats affichent :



Dans la section « compte actif », nous pouvons voir que *occupé* est l'état actif (le nombre actif est 1).

Conseils : Pour un State Composite, le nombre actif est de 1 (pour lui-même) *plus* le nombre de régions actives.

Dans la section « Pool d'événements », nous pouvons voir qu'il y a quatre occurrences d'événements dans la file d'attente d'événements. Chaque instance du signal transporte des données différentes.

L'ordre des événements dans la piscine est l'ordre dans lequel ils sont diffusés.

eval : Query Exécuter la valeur temporelle du contexte

Déclencheur AUTORISÉ,

[04817341] En attente du Déclencheur

[05494672] Commande : diffusion AUTORISÉE

[05494678] [serveur : TransactionServer] Événement mis en file d'attente : AUTORISÉ

[05494680] [server:TransactionServer] Événement distribué : AUTORISÉ

[05494686] [server:TransactionServer] Comportement de sortie : ServerStateMachine_busy

[05494686] [server:TransactionServer] **Effet de transition : occupé__TO__idle_61769**

[05494687] [server:TransactionServer] Comportement d'entrée : ServerStateMachine_idle

[05494688] [server:TransactionServer] Comportement d'exécution : ServerStateMachine_idle

[05495835] [server:TransactionServer] Achèvement : TransactionServer_ServerStateMachine_idle

[05495842] [server:TransactionServer] **Événement distribué : NEW_REQUEST.RequestSignal(requestType:3)**

[05495844] [server:TransactionServer] Comportement de sortie : ServerStateMachine_idle

[05495846] [server:TransactionServer] Effet de transition : ralenti__TO__busy_61772

[05495847] [server:TransactionServer] Comportement d'entrée : ServerStateMachine_busy

[05495850] [server:TransactionServer] Comportement d'exécution : ServerStateMachine_busy

[05496349] [server:TransactionServer] Achèvement : TransactionServer_ServerStateMachine_busy

[05496367] En attente du Déclencheur

- La transition de *occupé* à *inactif* est effectuée, nous nous attendons donc à ce que l'effet soit exécuté
- Un événement est rappelé du pool et distribué lorsque *l'inactivité* est terminée, ce qui fait que l'état *occupé* passe à l'état actif.
- Type *dump* et notez qu'il reste trois événements dans le pool ; le premier est rappelé et expédié

[05693348] Pool d'événements : [

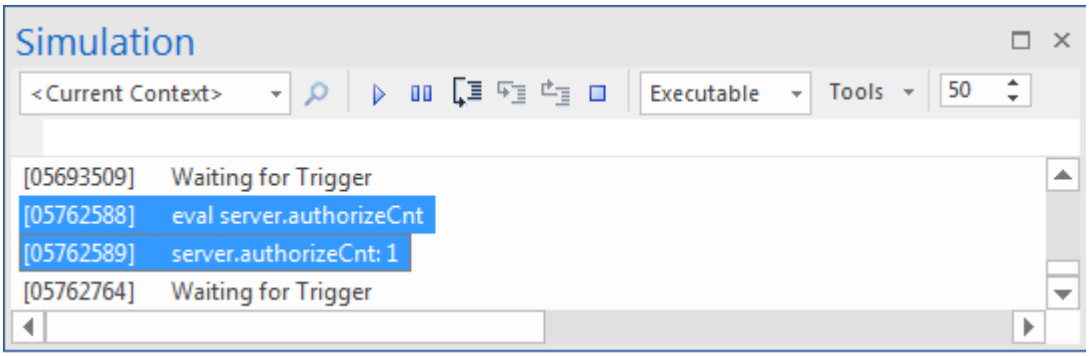
[05693349] NEW_REQUEST.RequestSignal(requestType:6),

[05693351] NEW_REQUEST.RequestSignal(requestType:5),

[05693352] NEW_REQUEST.RequestSignal(requestType:8),

[05693354]]

Type *eval server.authorizeCnt* dans le champ de texte. Ce chiffre indique que la valeur du temps exécuter de 'server.authorizeCnt' est 1.



Déclencheur AUTORISÉ à nouveau. Lorsque la Statemachine est stable à *occupé* , il restera deux événements dans le pool. Exécuter à nouveau *eval server.suthorizeCnt* ; la valeur sera 2.

Accéder à la variable membre du contexte à partir du comportement State et de l'effet de transition

Statemachine Exécutable d' Enterprise Architect supporte la simulation pour C, C++, C# , Java et JavaScript .

Pour C et C++, la syntaxe diffère de C# , Java et JavaScript pour accéder aux variables membres du contexte. C et C++ utilisent le pointeur '->' tandis que les autres utilisent simplement '.' ; cependant, vous pouvez toujours utiliser *this.variableName* pour accéder aux variables. Enterprise Architect le traduira par this->variableName pour C et C++.

Donc pour tous les langages, utilisez simplement ce format pour la simulation :

ceci.nomvariable

Exemples:

Dans l'effet de transition :

this.authorizeCnt++;

Dans le comportement d'entrée, d'action ou de sortie d'un état :

ceci.foo += ceci.bar;

Note : par défaut, Enterprise Architect remplace uniquement « this-> » par « this » pour C et C++ ; Par exemple:

this.foo = this.bar + monObject.iCount + monPointer->iCount ;

sera traduit en :

this->foo = this->bar + monObject.iCount + monPointer->iCount ;

Une liste complète des commandes prises en charge

Étant donné que l'artefact Statemachine Exécutable peut simuler plusieurs contextes ensemble, certaines commandes peuvent spécifier un nom d'instance.

exécuter Statemachine :

Comme chaque contexte peut avoir plusieurs Statemachines , la commande ' exécuter ' peut spécifier une Statemachine avec laquelle commencer.

- exécuter l'instance. statemachine
- exécuter all.all
- exécuter une instance
- exécuter tout
- exécuter

Par exemple:

exécuter

exécuter tout

exécuter le serveur

exécuter server.myMainStatemachine

diffuser et envoyer l'événement :

- diffuser une chaîne d'événements
- envoyer EventString à l'instance
- envoyer EventString (équivalent à diffuser EventString)

Par exemple:

diffuser l'événement 1

envoyer Event1 au client

Commande dump :

- décharge
- instance de vidage

Par exemple:

décharge

serveur de vidage

client de vidage

Commande d'évaluation :

- instance d'évaluation.nomvariable

Par exemple:

évaluer le client.requestCnt

serveur d'évaluation.responseCnt

Commande de sortie :

- sortie

Le format de l'EventString :

- EventName.SignalName (liste d'arguments)

Note : la liste d'arguments doit correspondre aux attributs définis dans le signal **par ordre** .

Par exemple, si le Signal définit deux attributs :

- foo
- bar

Alors ces EventStrings sont valides :

- Event1.Signal1(10, 5) --------- foo = 10; barre = 5
- Event1.Signal1(10,) --------- foo = 10; la barre n'est pas définie
- Event1.Signal1(,5) --------- barre = 10 ; foo n'est pas défini
- Event1.Signal1(,) --------- foo et bar ne sont pas définis

Si le Signal ne contient aucun attribut, nous pouvons simplifier le EventString pour :

- Nom de l'événement

Exemple : Entrer d'un State

La sémantique de l'entrée dans un State dépend du type d' State et de la manière dont on y entre.

Dans tous les cas, le comportement d'entrée de l' State est exécuté (si défini) dès l'entrée, mais seulement après que tout effet de comportement associé à la transition entrante soit terminé. De plus, si un comportement doActivity est défini pour le State , ce comportement commence son exécution immédiatement après l'exécution du comportement d'entrée.

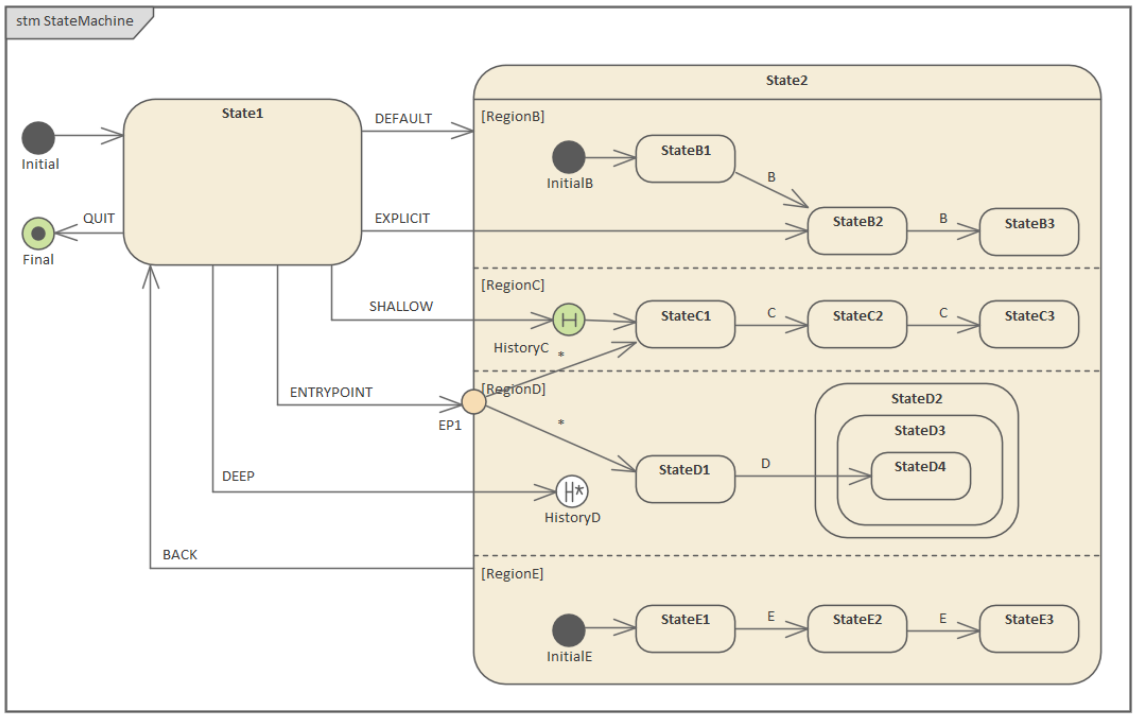
Pour un State composite avec une ou plusieurs régions définies, un certain nombre d'alternatives existent pour chaque région :

- *Entrée par défaut* : Cette situation se produit lorsque l' State composite propriétaire est la cible directe d'une transition ; après avoir exécuté l'entrée Behavior et effectué une éventuelle exécution de doActivity Behavior, l'entrée State continue à partir d'un pseudo-état initial via sa transition sortante (connue sous le nom de transition par défaut de l' State) si elle est définie dans la région.
  Si aucun pseudo-état initial n'est défini, cette région ne sera pas active
- *Entrée explicite* : si la transition entrante ou ses continuations se terminent sur un sous-état directement contenu de l' State composite propriétaire, alors ce sous-état devient actif et son comportement d'entrée est exécuté après l'exécution de l'entrée Comportement de l' State composite conteneur.
  Cette règle s'applique de manière récursive si la transition se termine sur un sous-état indirect (profondément imbriqué).
- *Entrée d'historique superficiel* : si la transition entrante se termine sur un pseudo-état d'histoire superficielle de cette région, le sous-état actif devient le sous-état qui était le plus récemment actif (sauf FinalState) avant cette entrée, sauf s'il s'agit de la première entrée dans cet State ; s'il s'agit de la première entrée dans cet State ou si l'entrée précédente a atteint une Finale, une transition historique superficielle par défaut sera prise si elle est définie, sinon l'entrée State par défaut est appliquée
- *Entrée d'historique profond* : la règle pour ce cas est la même que pour l'historique superficiel sauf que le pseudostate cible est de type deepHistory et que la règle est appliquée de manière récursive à tous les niveaux de la configuration State actif en dessous de celui-ci.
- *Entrée au point d'entrée* : Si une transition entre dans l' State composite propriétaire via un pseudo-état de point d'entrée, alors la transition sortante provenant du point d'entrée et pénétrant dans l' State dans cette région est prise en compte ; s'il y a plus de Transitions sortantes depuis les points d'entrée, chaque Transition doit cibler une Région différente et toutes les Régions sont activées simultanément

Pour States orthogonaux comportant plusieurs Régions, si la Transition entre explicitement dans une ou plusieurs Régions (dans le cas d'un Fork ou d'un point d'entrée), ces Régions sont renseignées explicitement et les autres par défaut.

Dans cet exemple, nous démontrons un modèle avec tous ces comportements d'entrée pour un State orthogonal.

Modélisation d'une Statemachine



Contexte de Statemachine

1. Créez un élément Class nommé *MyClass* , qui sert de contexte au Statemachine .
2. Cliquez-droit sur *MyClass* dans la fenêtre Navigateur et sélectionnez l'option 'Ajouter | Option Statemachine .

Statemachine

1. Ajoutez au diagramme un nœud *initial* , un State nommé *State1* , un State nommé *State2* et un élément Final nommé *final.*
2. Agrandissez *State2* sur le diagramme , cliquez-droit dessus et sélectionnez l'option 'Avancé | Définissez l'option des sous-états simultanés et définissez *RegionB, RegionC, RegionD* et *RegionE.*
3. Cliquez-droit sur *State2* et sélectionnez le 'New Child Element | Option Point d'entrée' pour créer le point d'entrée *EP1* .
4. Dans *RegionB* , créez les éléments *InitialB* , transition vers *StateB1* , transition vers *StateB2* , transition vers *StateB3* ; toutes les transitions déclenchées par l'événement *B.*
5. Dans *RegionC* , créez les éléments Shallow *HistoryC* (cliquez-droit sur le nœud History | Advanced | Deep History | décochez), transition vers *StateC1* , transition vers *StateC2* , transition vers *StateC3* ; toutes les transitions déclenchées par l'événement *C.*
6. Dans *RegionD* , créez les éléments deep *HistoryD* (cliquez-droit sur le nœud History | Advanced | Deep History | check), transition vers *StateD1* , créez *StateD2* comme parent de *StateD3* , qui est parent de *StateD4* ; passage de *StateD1* à *StateD4* ; déclenché par l'événement *D.*
7. Dans *RegionE* , créez les éléments *InitialE* , transition vers *StateE1* , transition vers *StateE2* , transition vers *StateE3* ; toutes les transitions déclenchées par l'événement *E.*
8. Dessinez des transitions du point d'entrée *EP1* vers *StateC1* et *StateD1.*

Dessinez des transitions pour différents types d'entrée :

1. Entrée par défaut : *State1* à *State2* ; déclenché par l'événement DEFAULT.
2. Entrée explicite : *State1* à *StateB2* ; déclenché par l'événement EXPLICIT.
3. Entrée d'historique peu profonde : *State1* à *HistoryC* ; déclenché par l'événement SHALLOW.
4. Entrée d'historique approfondi : *State1* à *HistoryD* ; déclenché par l'événement DEEP.
5. Entrée du point d'entrée : *State1* à *EP1* ; déclenché par l'événement ENTRYPOINT.

Autres transitions :

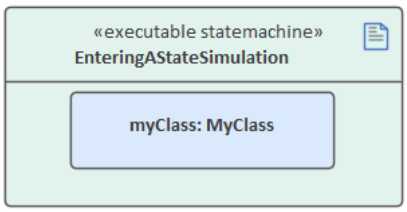
1. Sortie State composite : de *l'État 2* à *l'État 1* ; déclenché par l'événement BACK.
2. *State1* à *Final* , déclenché par l'événement QUIT.

Simulation

Artefact

Enterprise Architect supporte C, C++, C# , Java et JavaScript . Nous utilisons JavaScript dans cet exemple car nous n'avons pas besoin d'installer de compilateur. (Pour les autres langages, Visual Studio ou JDK sont requis.)

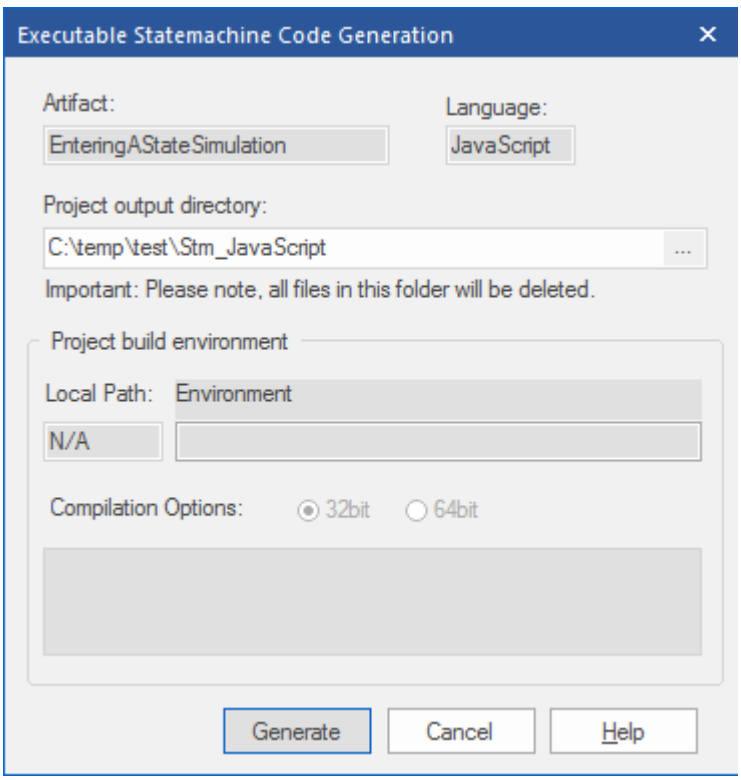
1. Sur la page ' Simulation ' de la boîte à outils Diagramme , faites glisser l'icône Statemachine Exécutable sur un diagramme et créez un artefact nommé *EnteringAStateSimulation* . Définissez le langage sur JavaScript .
2. Ctrl+faites glisser l'élément *MyClass* de la fenêtre Navigateur sur l'artefact *EnteringAStateSimulation* , sélectionnez l'option « Coller en tant que propriété » et donnez à la propriété le nom *myClass.*



Génération de code

1. Cliquez sur *EnteringAStateSimulation* et sélectionnez l'option de ruban 'Simulate > États Exécutables > Statemachine > Générer , Build and Exécuter '.
2. Spécifiez un répertoire pour le code source généré.

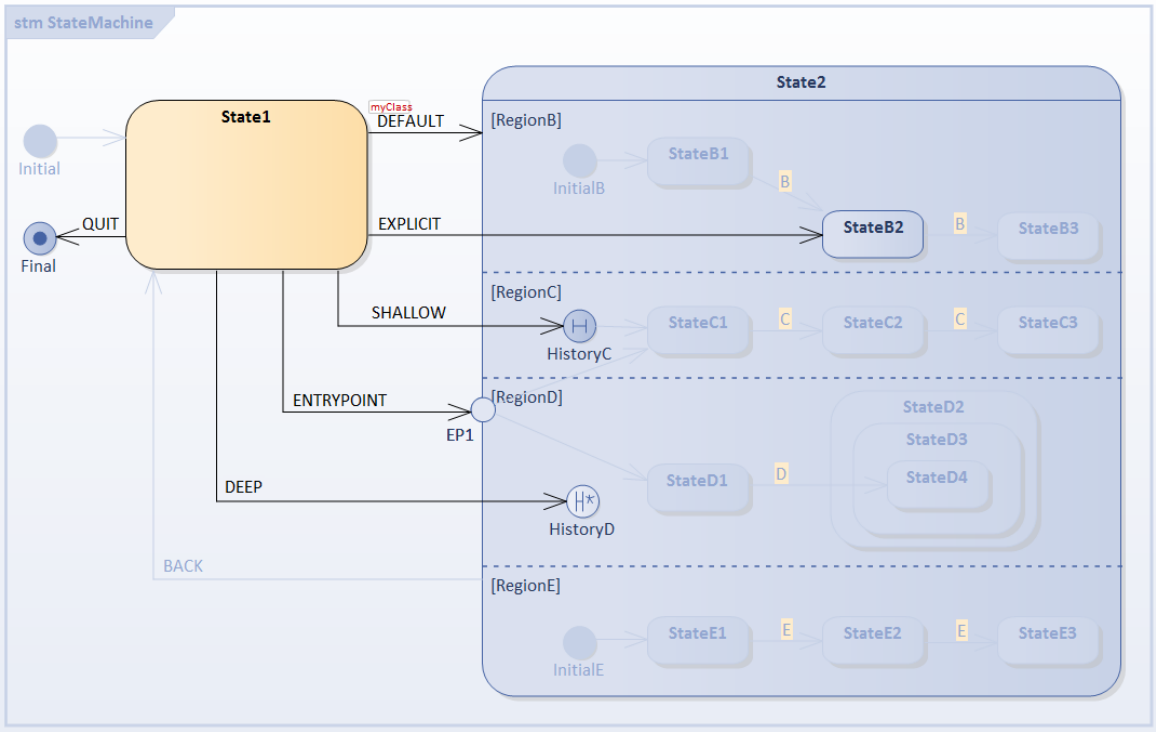
Note : Le contenu de ce répertoire sera effacé avant la génération ; assurez-vous de spécifier un répertoire utilisé uniquement à des fins de simulation Statemachine .



Exécuter Simulation

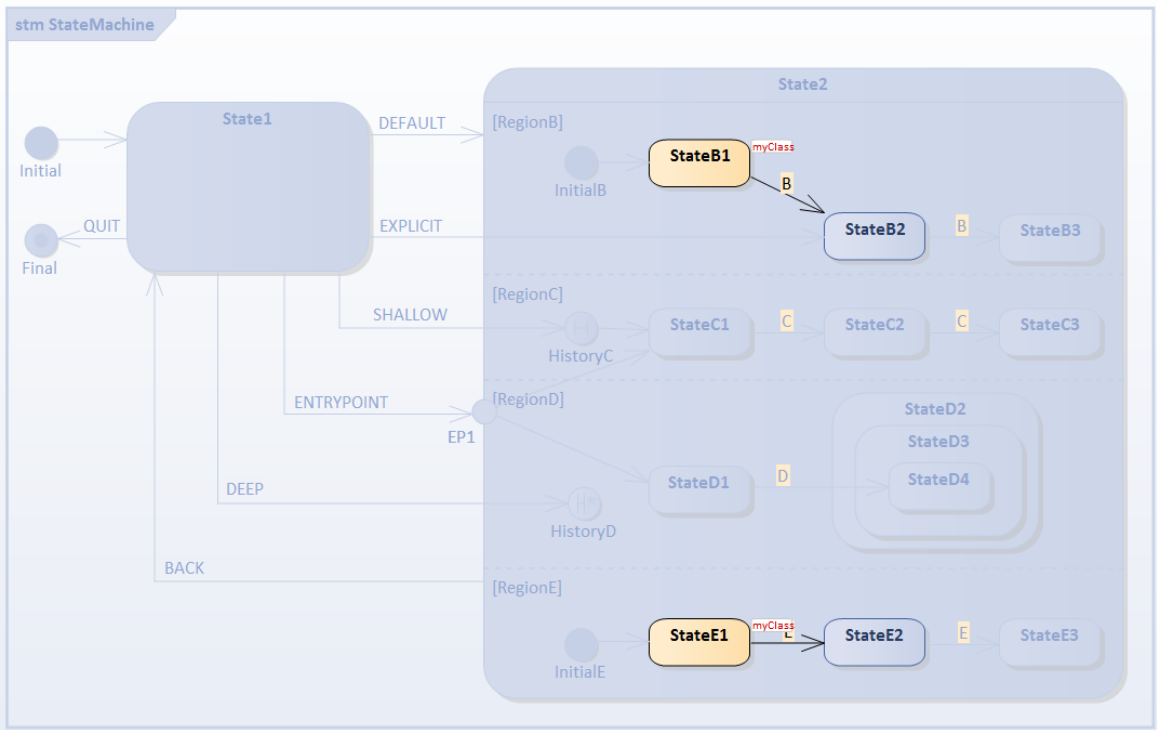
Conseils : Vous pouvez visualiser la séquence de trace d'exécution à partir de la fenêtre Simulation , que vous ouvrez en sélectionnant l'option de ruban 'Simuler > Simulation Dynamique > Simulateur > Ouvrir la fenêtre Simulation '.

Lorsque la simulation commence, *State1* est actif et Statemachine attend des événements.



Ouvrez la fenêtre Simulation Événements (Déclencheurs) à l'aide de l'option de ruban 'Simuler > Simulation Dynamique > Événements '.

1) Sélectionnez l'entrée par défaut : Déclencheur Séquence [DEFAULT].

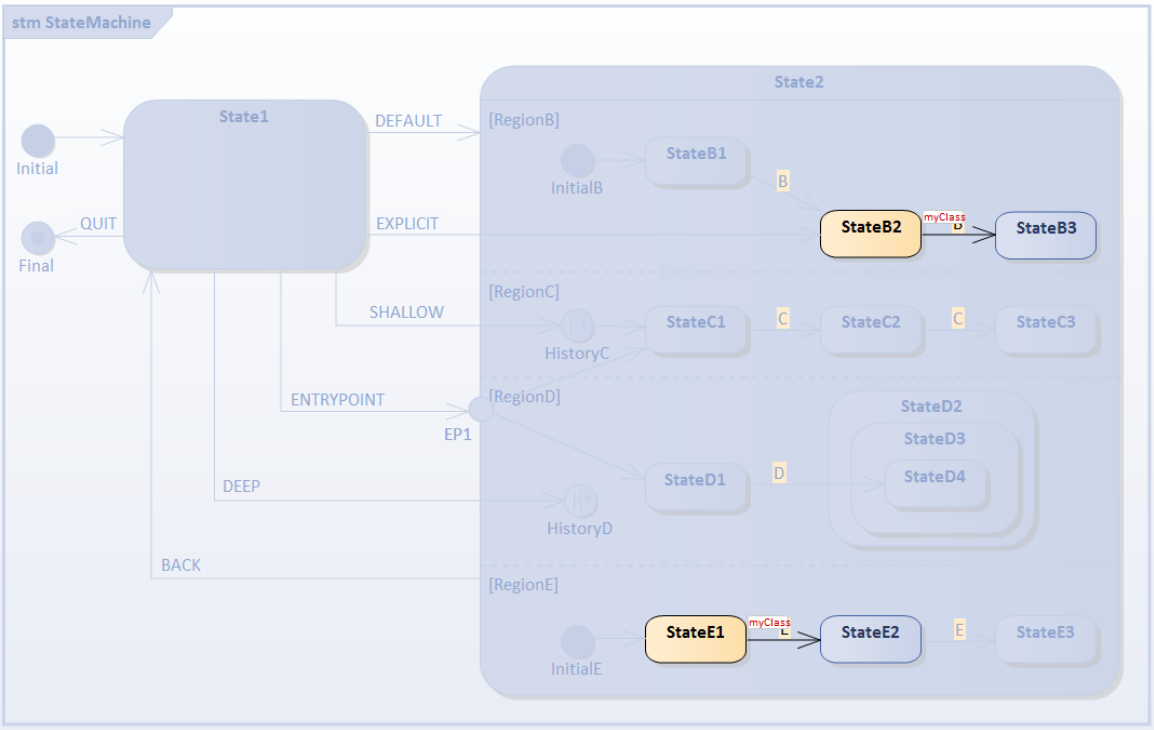


- *RegionB* est activé car il définit *InitialB* ; la transition qui en sort sera exécutée, *StateB1* est l'état actif

- *RegionE* est activé car il définit *InitialE* ; la transition qui en sort sera exécutée, *StateE1* est l'état actif
- *RegionC* et *RegionD* sont inactifs car aucun pseudo-état initial n'a été défini

Sélectionnez le Déclencheur [BACK] pour réinitialiser.

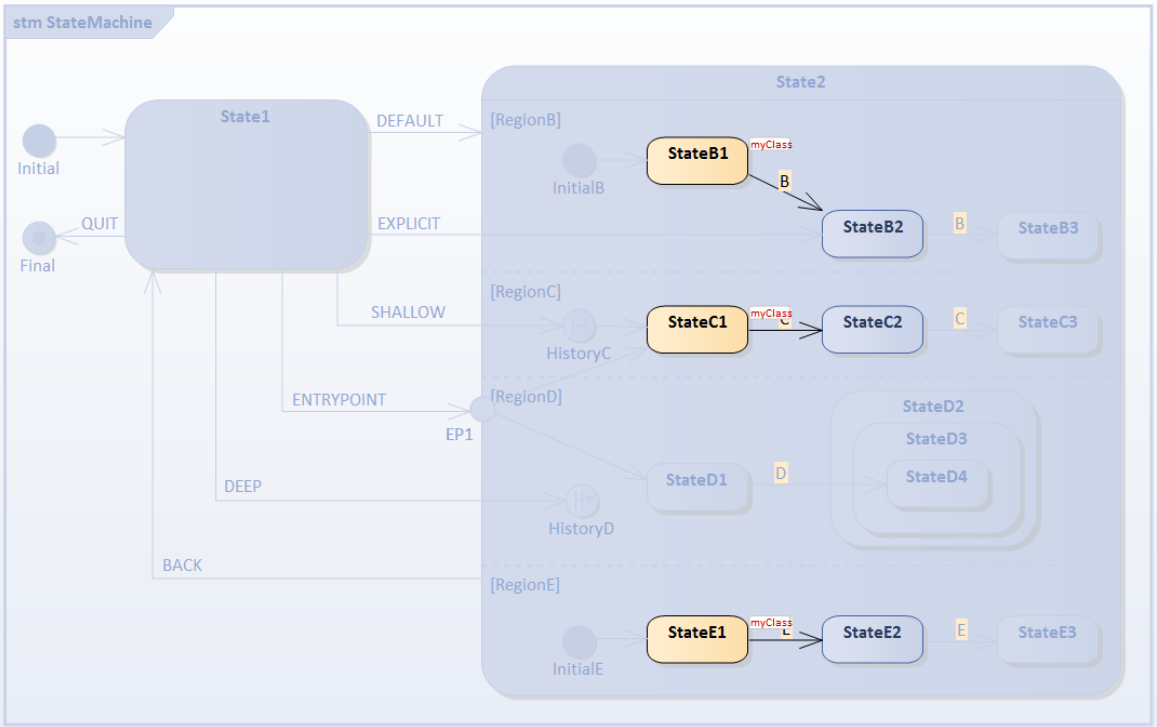
2) Sélectionnez l'entrée explicite : Déclencheur Séquence [EXPLICITE].



- *RegionB* est activé car la transition cible le sommet contenu *StateB2*
- *RegionE* est activé car il définit *InitialE* ; la transition qui en sort sera exécutée, *StateE1* est l'état actif
- *RegionC* et *RegionD* sont inactifs car aucun pseudo-état initial n'a été défini

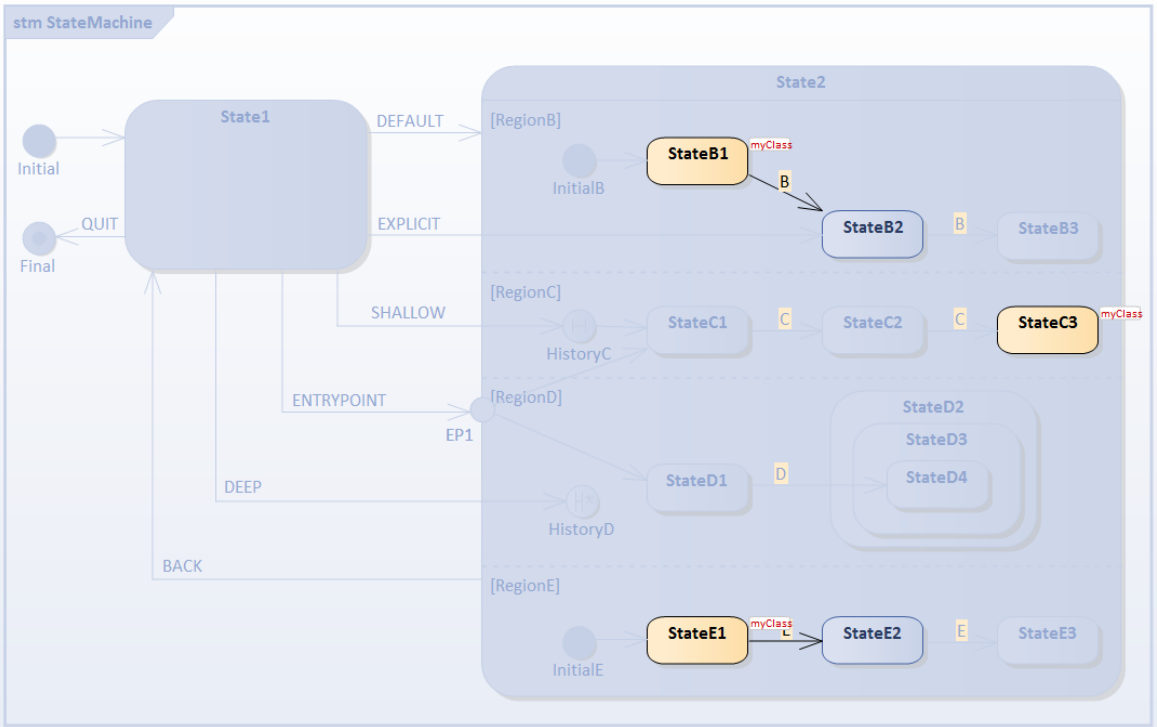
Sélectionnez le Déclencheur [BACK] pour réinitialiser.

3) Sélectionnez la transition historique par défaut : Déclencheur Séquence [SHALLOW].



- *RegionC* est activé car la transition cible le sommet contenu *HistoryC* ; puisque cette région est entrée pour la première fois (et que le pseudo-état History n'a rien à « retenir »), la transition sortant de *HistoryC* vers *StateC1* est exécutée
- *RegionB* est activé car il définit *InitialB* ; la transition qui en sort sera exécutée, *StateB1* est l'état actif
- *RegionE* est activé car il définit *InitialE* ; la transition qui en sort sera exécutée, *StateE1* est l'état actif
- *RegionD* est inactif car aucun pseudo-état initial n'a été défini

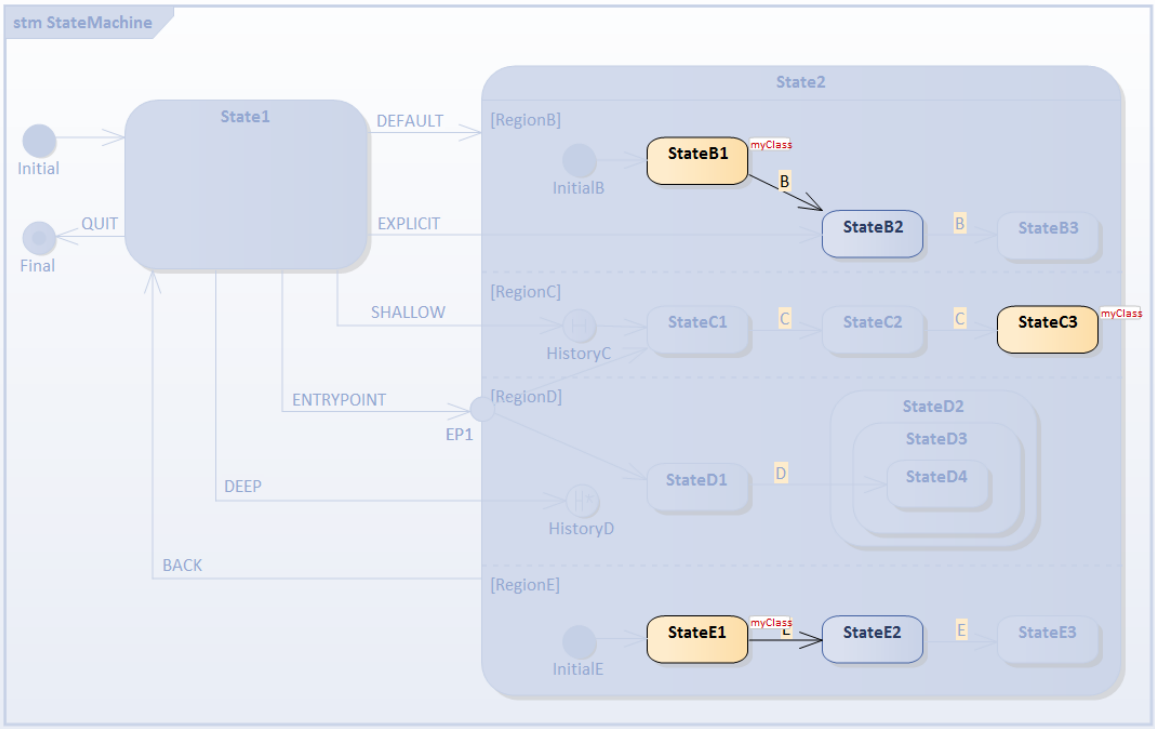
4) Préparez-vous au test de l'entrée d'historique superficielle : Séquence Déclencheur [C, C].



- Nous supposons que le pseudo-état historique superficiel *HistoryC* peut se souvenir *de StateC3*

Sélectionnez le Déclencheur [BACK] pour réinitialiser.

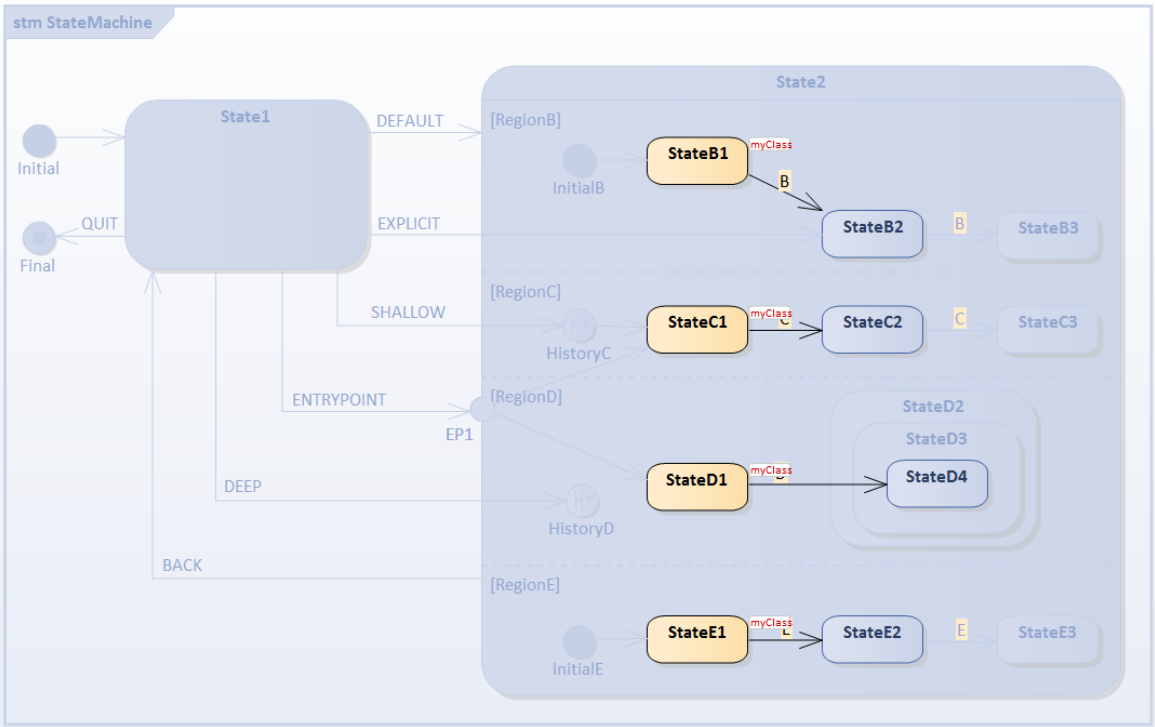
5) Sélectionnez l'entrée d'historique peu profonde : Déclencheur Séquence [SHALLOW].



- Pour *RegionC* , *StateC3* est activé directement

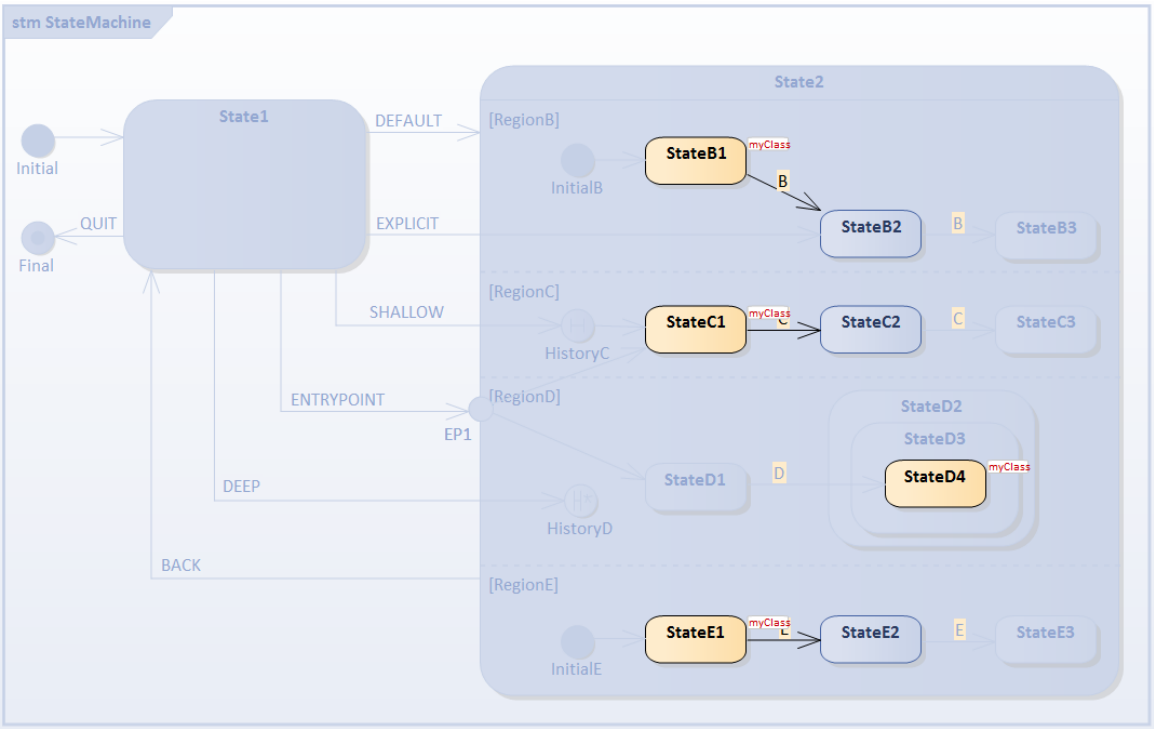
Sélectionnez le Déclencheur [BACK] pour réinitialiser.

6) Sélectionnez l'entrée du point d'entrée : Séquence Déclencheur [ENTRYPOINT].



- *RegionC* est activé car la transition depuis *EP1* cible le *StateC1* contenu
- *RegionD* est activé car la transition depuis *EP1* cible le *StateD1* contenu
- *RegionB* est activé car il définit *InitialB* ; la transition qui en sort sera exécutée, *StateB1* est l'état actif
- *RegionE* est activé car il définit *InitialE* ; la transition qui en sort sera exécutée, *StateE1* est l'état actif

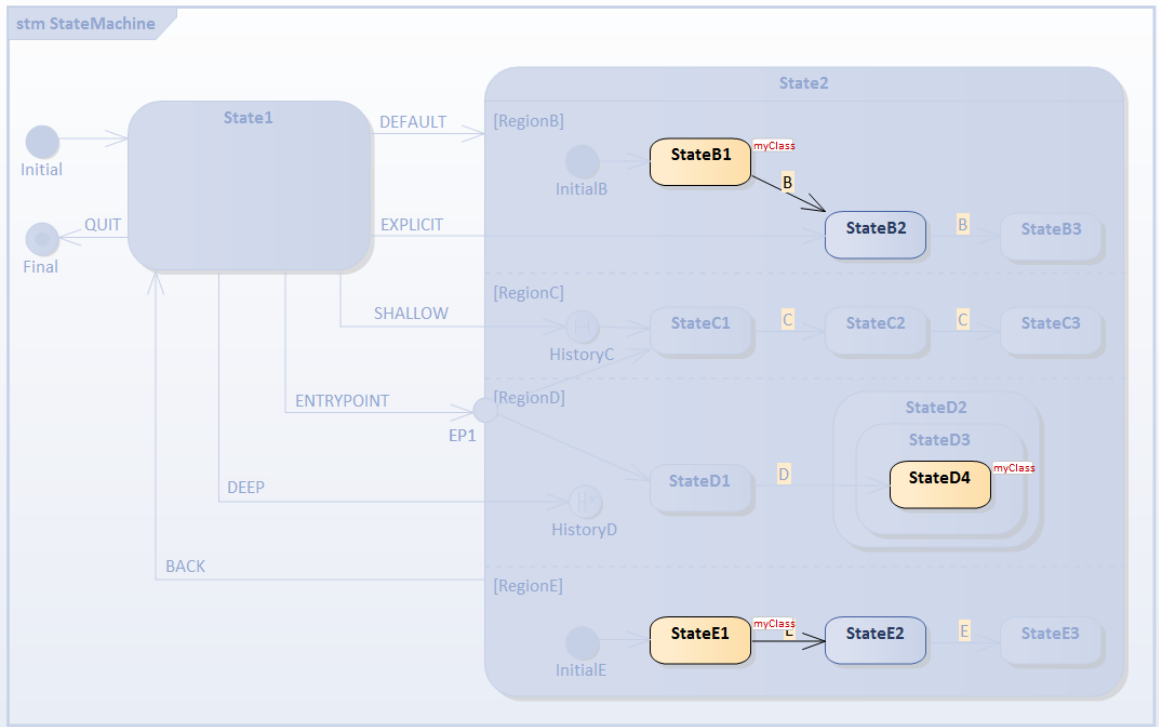
7) Préparez-vous à tester Deep History : Déclencheur Séquence [D].



- Nous supposons que le pseudo-état d'histoire profonde *HistoryD* peut se souvenir *de StateD2* , *StateD3* et *StateD4.*

Sélectionnez le Déclencheur [BACK] pour réinitialiser.

8) Sélectionnez l'entrée de l'historique profond : Déclencheur Séquence [DEEP].



- Pour *RegionD* , *StateD2* , *StateD3* et *StateD4* sont saisis ; les traces sont :
  - maClasse[MaClasse].StateMachine_State1 QUITTER

- myClass[MyClass].State1__TO__HistoryD_105793_61752 Effet

- maClasse[MaClasse].StateMachine_State2 ENTRÉE

- maClasse[MaClasse].StateMachine_State2 DO

- myClass[MyClass].InitialE_105787__TO__StateE1_61746 Effet
- maClasse[MaClasse].StateMachine_State2_StateE1 ENTRÉE
- maClasse[MaClasse].StateMachine_State2_StateE1 DO
- myClass[MyClass].InitialB_105785__TO__StateB1_61753 Effet
- maClasse[MaClasse].StateMachine_State2_StateB1 ENTRÉE
- maClasse[MaClasse].StateMachine_State2_StateB1 DO
- maClasse[MaClasse].StateMachine_State2_StateD2 ENTRÉE
- maClasse[MaClasse].StateMachine_State2_StateD2_StateD3 ENTRÉE
- maClasse[MaClasse].StateMachine_State2_StateD2_StateD3_StateD4 ENTRÉE

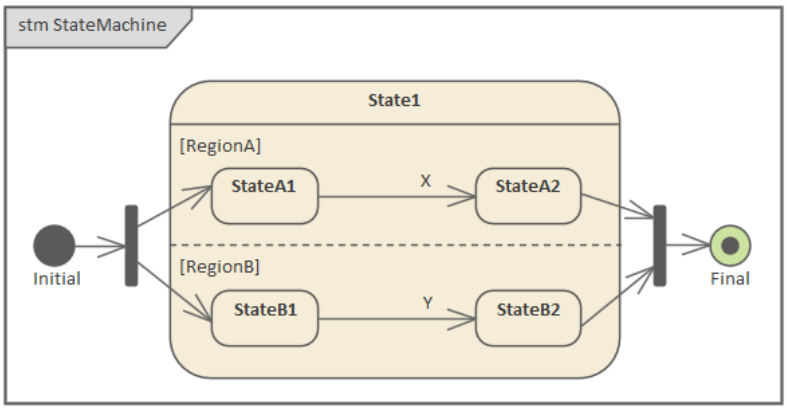
Exemple : Fourche et Joindre

Les pseudo-états de fourche divisent une transition entrante en deux ou plusieurs transitions, se terminant par des sommets dans des régions orthogonales d'un State composite. Les Transitions sortant d'un pseudo-état Fork ne peuvent pas avoir de garde ou de déclencheur , et les comportements d'effet des Transitions sortantes individuelles sont, au moins conceptuellement, exécutés simultanément.

Les pseudo-états de jointure sont un sommet cible commun pour deux ou plusieurs transitions provenant de sommets dans différentes régions orthogonales. Les pseudo-états de jointure remplissent une fonction de synchronisation, dans laquelle toutes les transitions entrantes doivent se terminer avant que l'exécution puisse continuer via une transition sortante.

Dans cet exemple, nous démontrons le comportement d'une Statemachine avec les pseudo-états Fourche et Joindre .

Modélisation Statemachine



Contexte de Statemachine

- Créez un élément Class nommé *MyClass* , qui sert de contexte à une Statemachine
- Cliquez-droit sur *MyClass* dans la fenêtre Navigateur et sélectionnez l'option 'Ajouter | Option Statemachine

Statemachine

- Ajoutez un *Initial* Node, un *Fork* , un State nommé *State1* , un *Join* et un *Final* au diagramme
- Agrandir *l'état1* , cliquez-droit dessus sur le diagramme et sélectionnez l'option 'Avancé | Définir des sous-états simultanés | Définir l'option et définir *RegionA* et *RegionB*
- Dans *RegionA* , définissez *StateA1* , transition vers *StateA2* , déclenchée par l'événement *X*
- Dans *RegionB* , définissez *StateB1* , transition vers *StateB2* , déclenchée par l'événement *Y*
- Dessinez d'autres transitions : *Initiale* à *Fork* ; *Fork* vers *StateA1* et *StateB1* ; *StateA2 et StateB2* à *rejoindre* ; *Rejoindre* la *finale*

Simulation

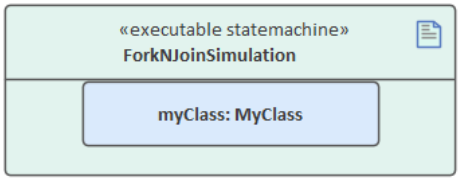
Artefact

Enterprise Architect supporte C, C++, C# , Java et JavaScript ; nous utiliserons JavaScript dans cet exemple car nous n'avons pas besoin d'installer de compilateur (pour les autres langages, Visual Studio ou JDK sont requis).

- Dans la boîte à outils Diagramme , sélectionnez la page « Simulation » et faites glisser l'icône Statemachine Exécutable sur le diagramme pour créer un artefact ; nommez-le *ForkNJoinSimulation* et définissez son champ

'Langue' sur ' JavaScript '

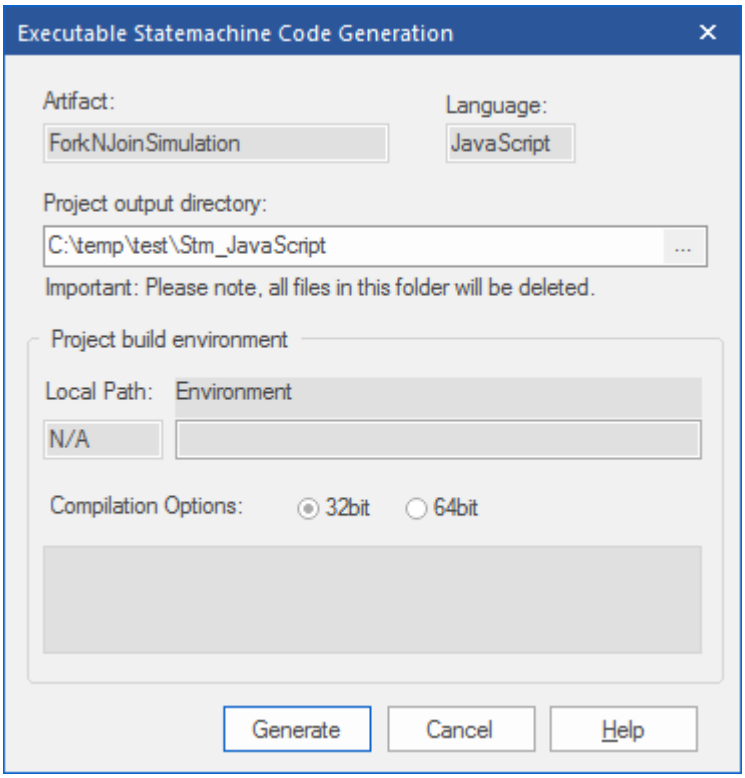
- Ctrl+Faites glisser *MyClass* depuis la fenêtre Navigateur et déposez-la sur l'artefact *ForkNJoinSimulation* en tant que propriété ; donnez-lui le nom *maClasse*



Génération de code

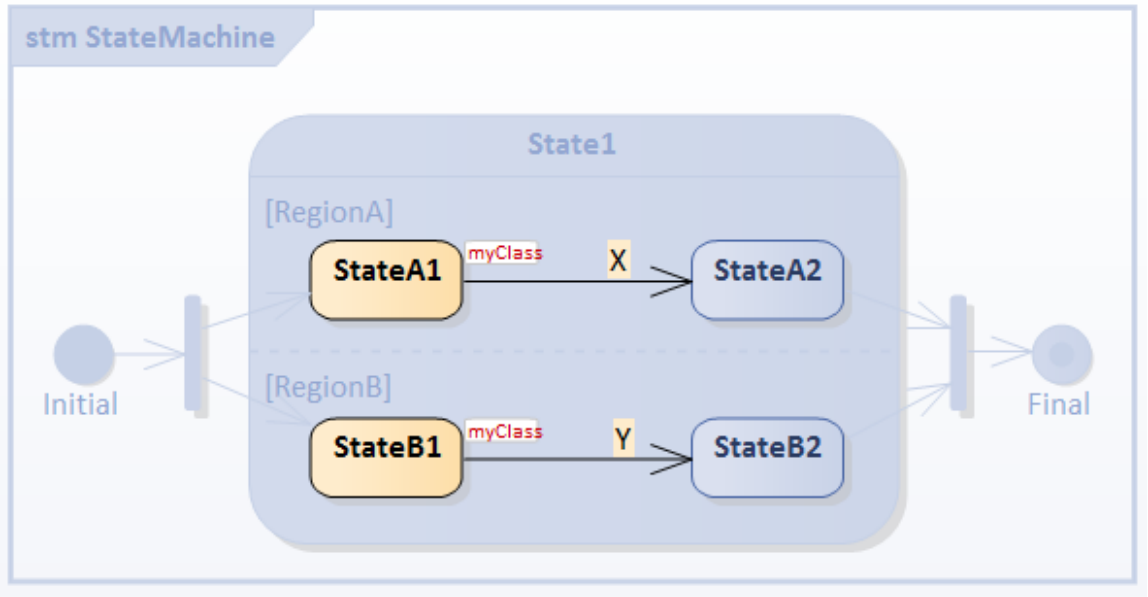
- Cliquez sur *ForkNJoinSimulation* et sélectionnez l'option de ruban 'Simulate > États Exécutables > Statemachine > Générer , Build and Exécuter '
- Spécifiez un répertoire pour le code source généré

Note : Le contenu de ce répertoire sera effacé avant la génération ; assurez-vous de pointer vers un répertoire qui existe uniquement à des fins de simulation Statemachine .



Exécuter Simulation

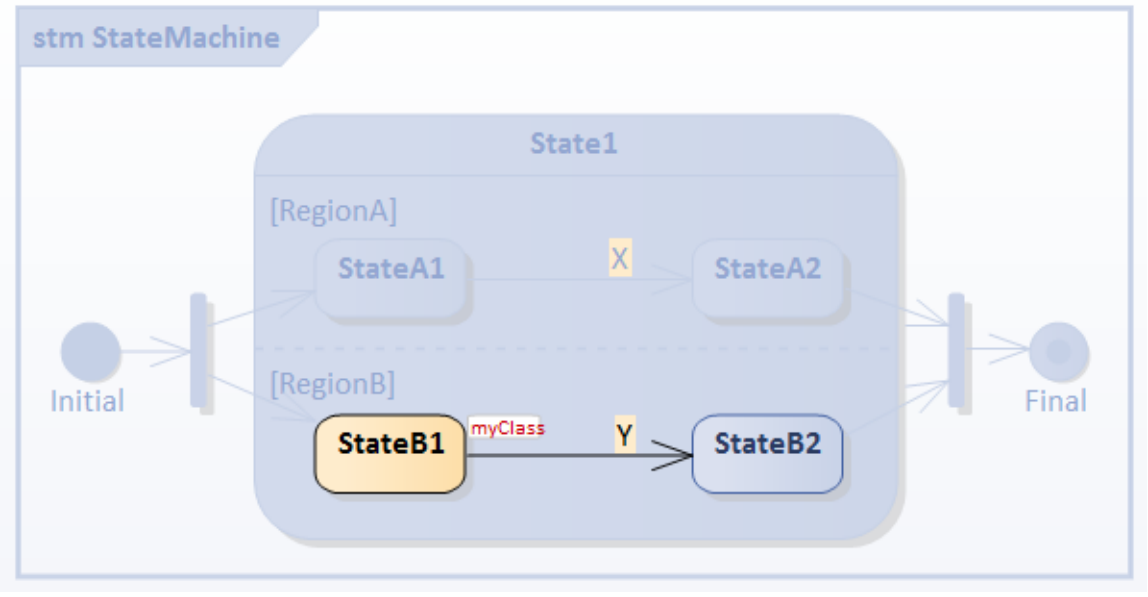
Lorsque la simulation est démarrée, *State1* , *StateA1* et *StateB1* sont actifs et la Statemachine attend des événements.



Sélectionnez l'option de ruban 'Simuler > Simulation Dynamique > Événements ' pour afficher la fenêtre Simulation Événements .

Lors de l'événement Déclencheur *X* , *StateA1* sortira et entrera dans *StateA2* ; une fois que le comportement Entry et doActivity a été exécuter , les événements d'achèvement de *StateA2* sont distribués et rappelés. Ensuite, la transition de *StateA2* au pseudo-état *Join* est activée et parcourue.

Note : *Join* doit attendre que toutes les transitions entrantes soient terminées avant que l'exécution puisse continuer via une transition sortante. Puisque le branchement depuis *RegionB* n'est pas terminé (car *StateB1* est toujours actif et en attente de déclencheurs) la transition de *Join* vers *Final* ne sera pas exécutée pour le moment.



Lors de l'événement Déclencheur *Y* , *StateB1* sortira et entrera dans *StateB2* ; une fois que le comportement d'entrée et de doActivity a été exécuter , les événements d'achèvement de *StateB2* sont distribués et rappelés. Ensuite, la transition de *StateB2* vers le pseudo-état *Join* est activée et parcourue. Cela satisfait aux critères de toutes les transitions entrantes de *Join* étant terminées, donc la transition de *Join* à *Final* est exécutée. Simulation est terminée.

Conseils : Vous pouvez visualiser la séquence de trace d'exécution depuis la fenêtre Simulation (option de ruban 'Simuler > Simulation Dynamique > Simulateur > Ouvrir la fenêtre Simulation ').

maClasse[MaClasse].Initial_82285__TO__fork_82286_82286_61745 Effet

maClasse[MaClasse].StateMachine_State1 ENTRY
maClasse[MaClasse].StateMachine_State1 DO
maClasse[MaClasse].fork_82286_82286__TO__StateA1_57125 Effet
maClasse[MaClasse].StateMachine_State1_StateA1 ENTRÉE
maClasse[MaClasse].StateMachine_State1_StateA1 DO
maClasse[MaClasse].fork_82286_82286__TO__StateB1_57126 Effet
maClasse[MaClasse].StateMachine_State1_StateB1 ENTRÉE
maClasse[MaClasse].StateMachine_State1_StateB1 DO
Déclencheur X
maClasse[MaClasse].StateMachine_State1_StateA1 QUITTER
maClasse[MaClasse].StateA1__TO__StateA2_57135 Effet
maClasse[MaClasse].StateMachine_State1_StateA2 ENTRÉE
maClasse[MaClasse].StateMachine_State1_StateA2 DO
maClasse[MaClasse].StateMachine_State1_StateA2 QUITTER
myClass[MyClass].StateA2__TO__join_82287_82287_57134 Effet
Déclencheur Y
maClasse[MaClasse].StateMachine_State1_StateB1 QUITTER
maClasse[MaClasse].StateB1__TO__StateB2_57133 Effet
maClasse[MaClasse].StateMachine_State1_StateB2 ENTRÉE
maClasse[MaClasse].StateMachine_State1_StateB2 DO
maClasse[MaClasse].StateMachine_State1_StateB2 QUITTER
maClasse[MaClasse].StateB2__TO__join_82287_82287_57132 Effet
maClasse[MaClasse].StateMachine_State1 QUITTER
maClasse[MaClasse].join_82287_82287__TO__Final_105754_57130 Effet

Exemple : Pseudo-state Historique

L'historique State est un concept pratique associé aux régions des States composites, dans lequel une région garde une trace de la configuration dans laquelle se trouvait un State lors de sa dernière sortie. Cela permet de revenir facilement à cette configuration State , si nécessaire, lorsque la région devient ensuite active (par exemple, après le retour de la gestion d'une interruption), ou s'il existe une transition locale qui revient à son historique.

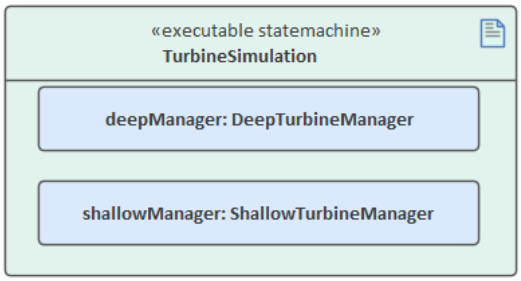
Enterprise Architect supporte deux types de Pseudo-state Historique :

- Historique approfondi - représentant la configuration complète de State de la visite la plus récente dans la région contenante ; l'effet est le même que si la transition se terminant sur le pseudo-état deepHistory s'était terminée sur l' State le plus interne de la configuration de l' State préservé, y compris l'exécution de tous les comportements d'entrée rencontrés en cours de route.
- Historique superficiel : représente un retour uniquement au sous-état le plus élevé de la configuration State la plus récente, qui est saisie à l'aide de la règle de saisie par défaut.

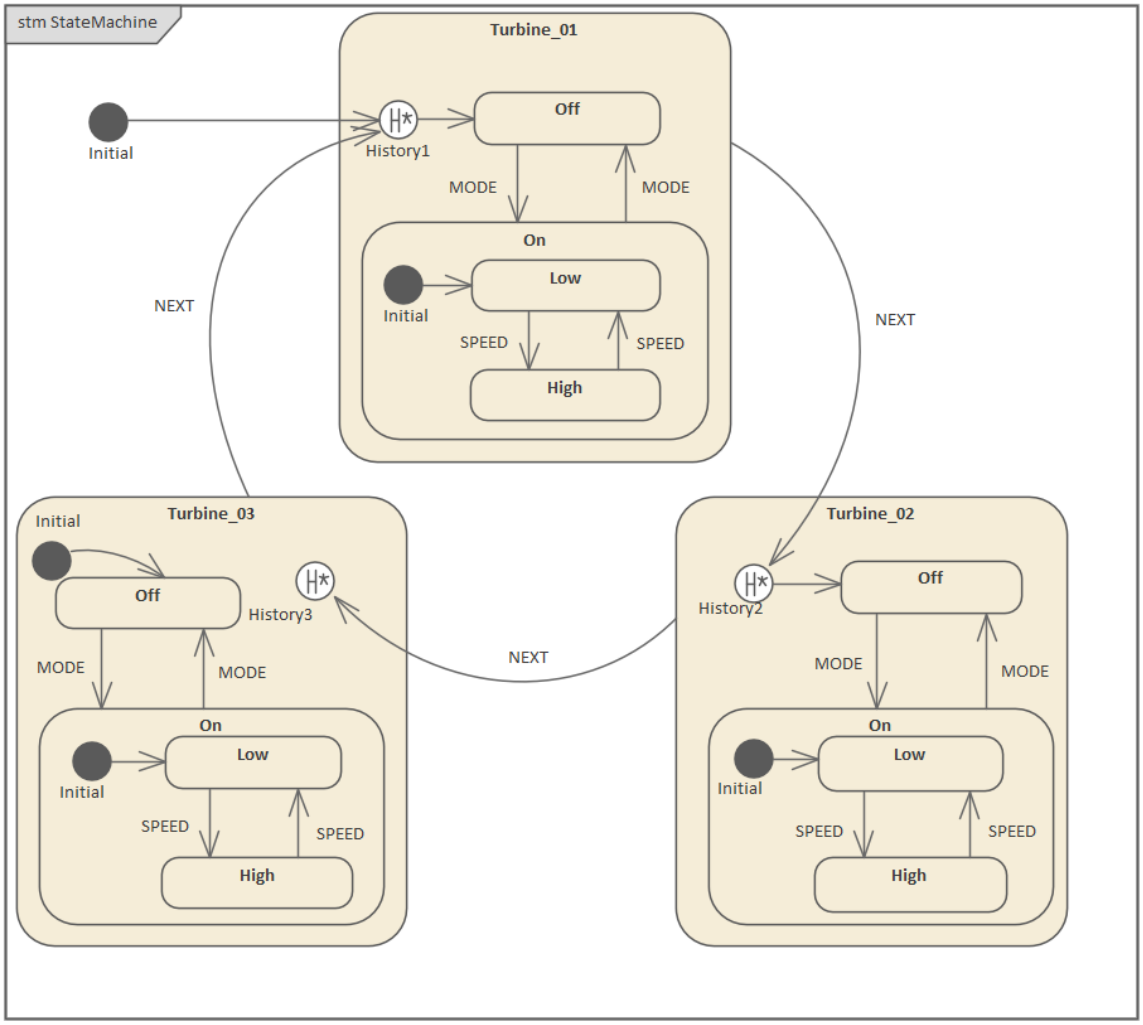
Dans cet exemple, les classes *DeepTurbineManager* et *ShallowTurbineManager* sont exactement les mêmes, sauf que la Statemachine contenue pour la première a un pseudo-état deepHistory et pour la seconde, un pseudo-état ShallowHistory.

Les deux Statemachines ont trois States composites : *Turbine_01* , *Turbine_02* et *Turbine_03* , dont chacun a States *Off* et *On* et un Pseudo-state Historique dans sa région.

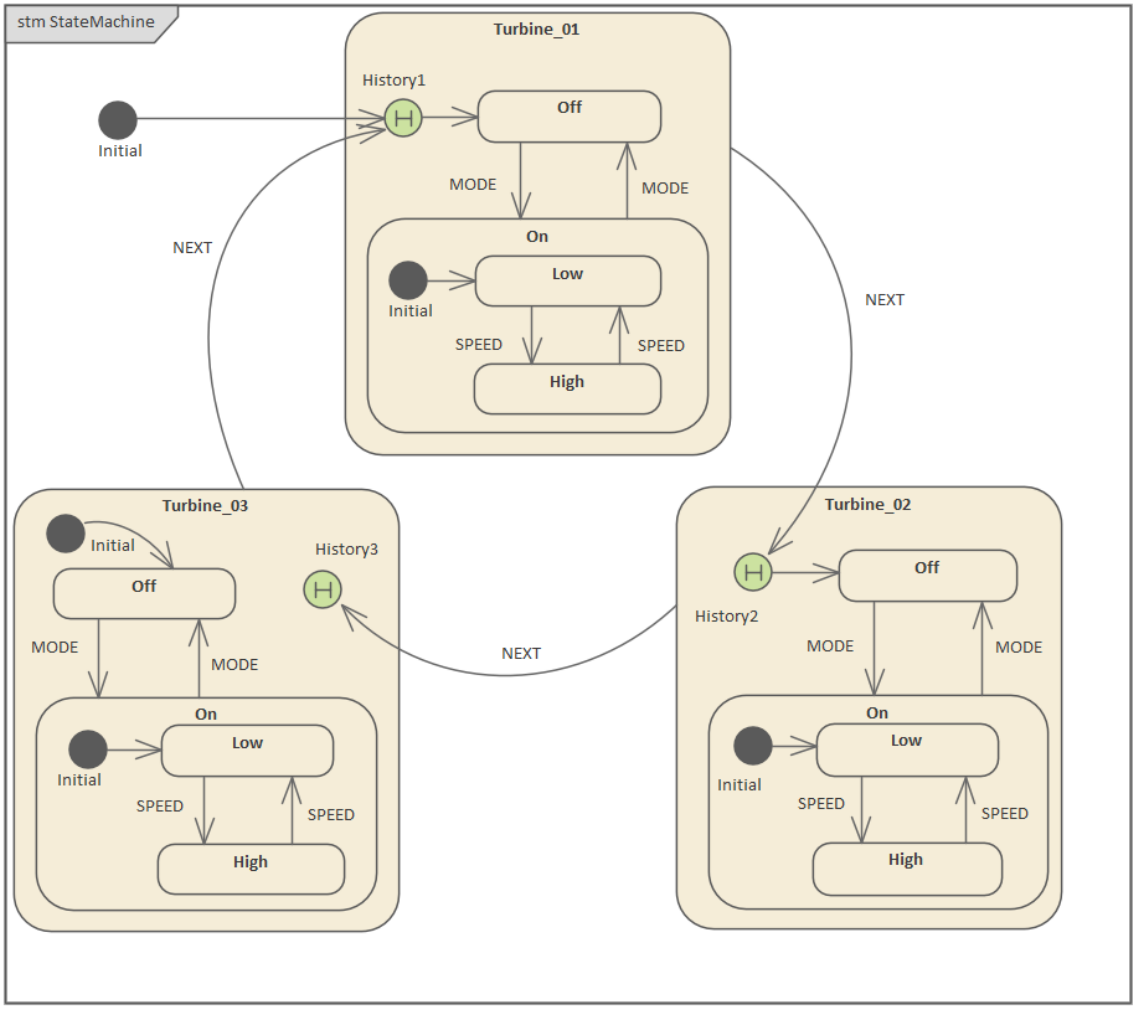
Afin de mieux observer la différence entre Deep History et Shallow History, nous exécutons les deux Statemachines dans une seule simulation.



La Statemachine dans *DeepTurbineManager* est illustrée dans ce diagramme :



La Statemachine dans *ShallowTurbineManager* est illustrée dans ce diagramme :

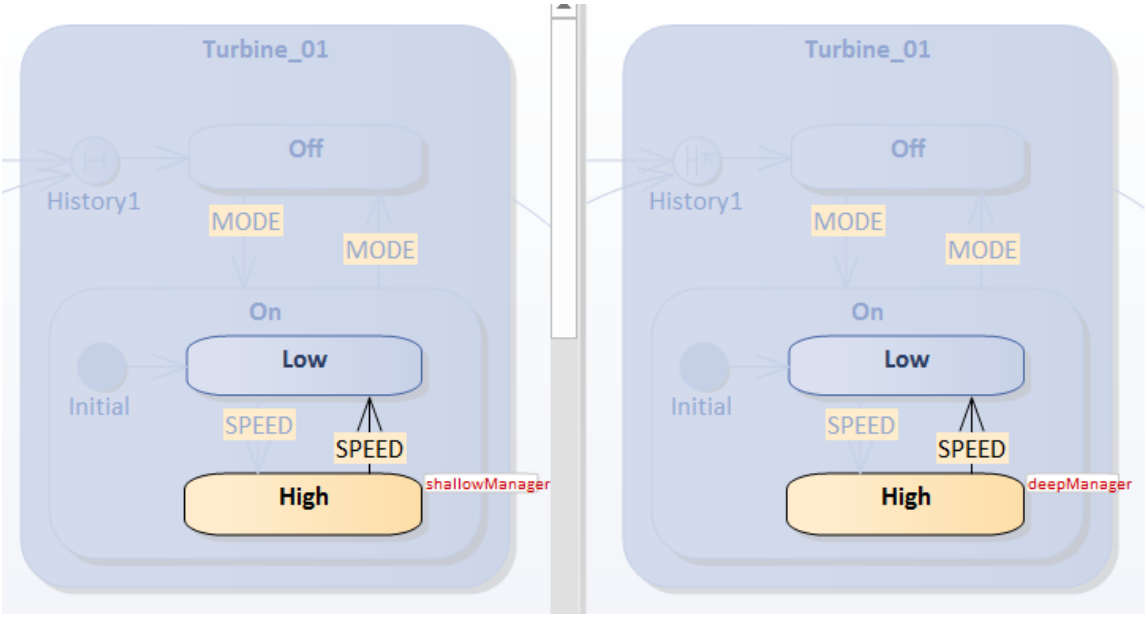


Conseil : Si vous cliquez-droit sur le noeud Historique du diagramme et sélectionnez l'option 'Avancé | Dans l'option Deep History, vous pouvez basculer le type de Pseudo-state Historique entre superficiel et profond.

Première activation des States

Après le démarrage de la simulation, *Turbine_01* et son sous-état *Off* sont activés.

Séquence Déclencheur : [MODE, SPEED]

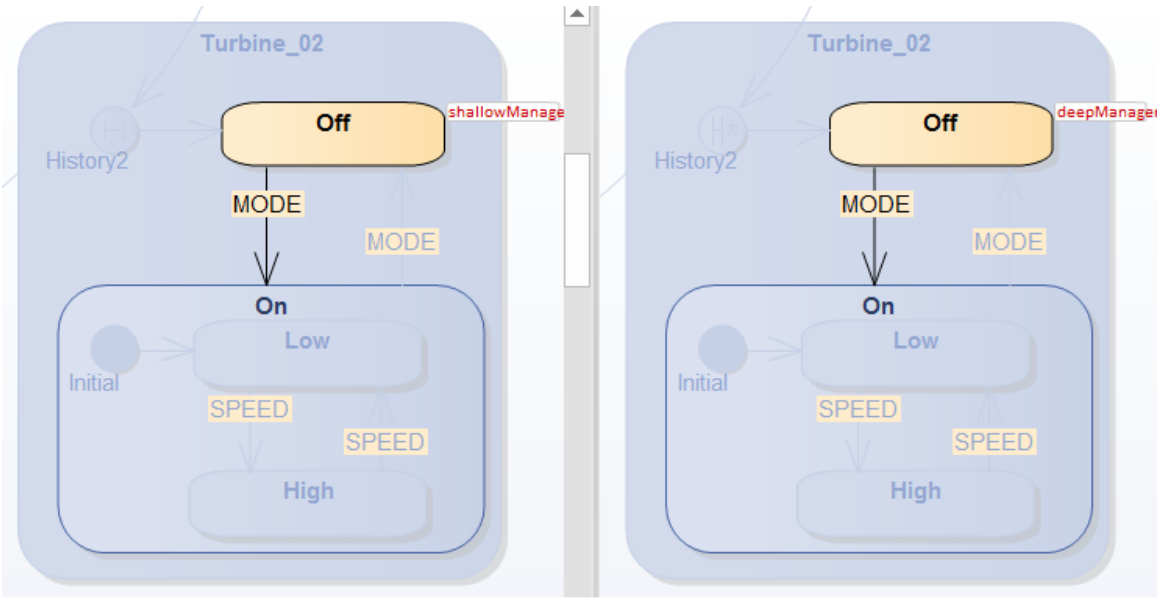


Ensuite, la configuration State actif comprend :

- Turbine_01
- Turbine_01.Activée
- Turbine_01.On.High

Cela s'applique à la fois à *deepManager* et *àshallowManager.*

Séquence Déclencheur : [SUIVANT]



Cette séquence de traces peut être observée depuis la fenêtre Simulation (Simuler > Simulation Dynamique > Simulateur > Ouvrir la fenêtre Simulation) :

01shallowManager[ShallowTurbineManager].StateMachine_Turbine_01_On_High EXIT

02shallowManager[ShallowTurbineManager].StateMachine_Turbine_01_On EXIT

03 ShallowManager[ShallowTurbineManager].StateMachine_Turbine_01 QUITTER

04 ShallowManager[ShallowTurbineManager].Turbine_01__TO__History2_105720_61730 Effet

05 ShallowManager[ShallowTurbineManager].StateMachine_Turbine_02 ENTRÉE

06shallowManager[ShallowTurbineManager].StateMachine_Turbine_02 DO

07 ShallowManager[ShallowTurbineManager].History2_105720__TO__Off_61731 Effet

08 ShallowManager[ShallowTurbineManager].StateMachine_Turbine_02_Off ENTRÉE

09 ShallowManager[ShallowTurbineManager].StateMachine_Turbine_02_Off DO

Note : Puisque *deepManager* a exactement la même trace que *ShallowManager* , la trace de *deepManager* est filtrée de cette séquence.

On peut apprendre que :

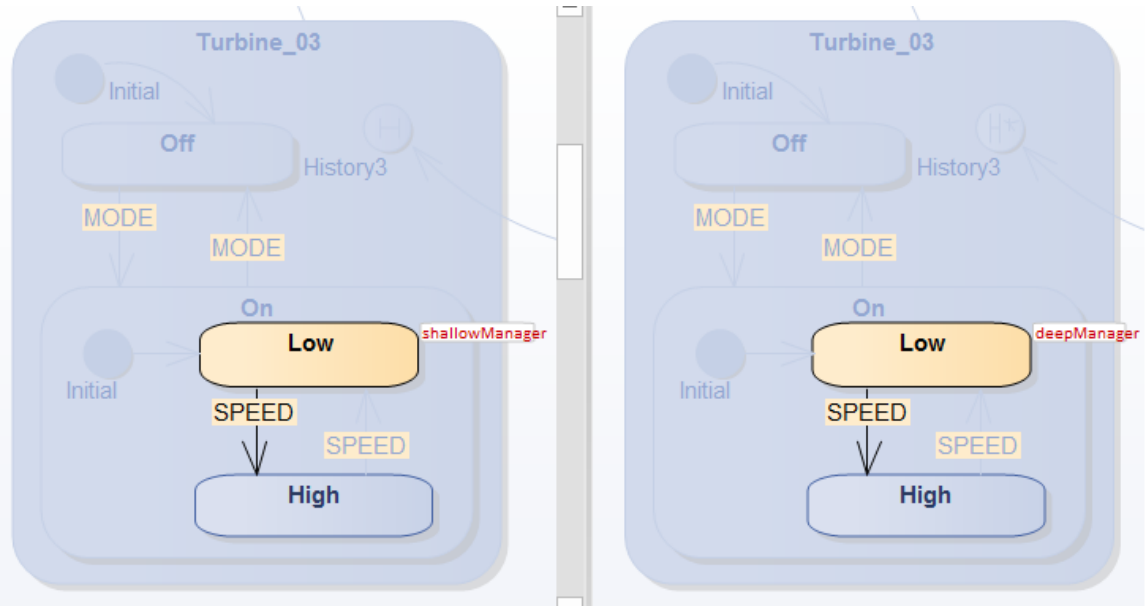
- La sortie d'un State composite commence par l' State le plus interne dans la configuration State actif (voir les lignes 01 à 03 dans la séquence de trace)
- La transition historique par défaut n'est prise que si l'exécution mène au nœud Historique (voir ligne 04) et que l' State n'a jamais été actif auparavant (voir ligne 07).

Ensuite, la configuration State actif comprend :

- Turbine_02
- Turbine_02.Désactivé

Cela s'applique à la fois à *deepManager* et *àshallowManager.*

Séquence Déclencheur : [NEXT, MODE]



Cette séquence de traces peut être observée depuis la fenêtre Simulation :

Déclencheur [SUIVANT]

01shallowManager[ShallowTurbineManager].StateMachine_Turbine_02_Off QUITTER

02 ShallowManager[ShallowTurbineManager].StateMachine_Turbine_02 QUITTER

03 ShallowManager[ShallowTurbineManager].Turbine_02__TO__History3_105713_61725 Effet

04 ShallowManager[ShallowTurbineManager].StateMachine_Turbine_03 ENTRÉE

05shallowManager[ShallowTurbineManager].StateMachine_Turbine_03 DO

06 ShallowManager[ShallowTurbineManager].Initial_105706__TO__Off_61718 Effet

07 ShallowManager[ShallowTurbineManager].StateMachine_Turbine_03_Off ENTRÉE

08shallowManager[ShallowTurbineManager].StateMachine_Turbine_03_Off DO

Déclencheur [MODE]

Message omis...

Note : Puisque *deepManager* a exactement la même trace que *ShallowManager,* la trace de *deepManager* est filtrée de cette séquence.

On peut apprendre que :

- Puisqu'il n'y a pas de transition historique par défaut définie pour *History3* , l'entrée par défaut standard de l' State est effectuée ; un nœud Initial est trouvé dans la Région contenue par *Turbine_03,* donc la Transition issue de *Initial* est activée (voir ligne 06)

Ensuite, la configuration de l'état actif comprend :

- Turbine_03
- Turbine_03.Activée
- Turbine_03.On.Low

Cela s'applique à la fois à *deepManager* et *àshallowManager.*

Entrée dans l'histoire des States

À titre de référence, nous montrons l'instantané Deep History de chaque turbine après sa première activation :

Turbine_01

- Turbine_01.Activée
- Turbine_01.On.High

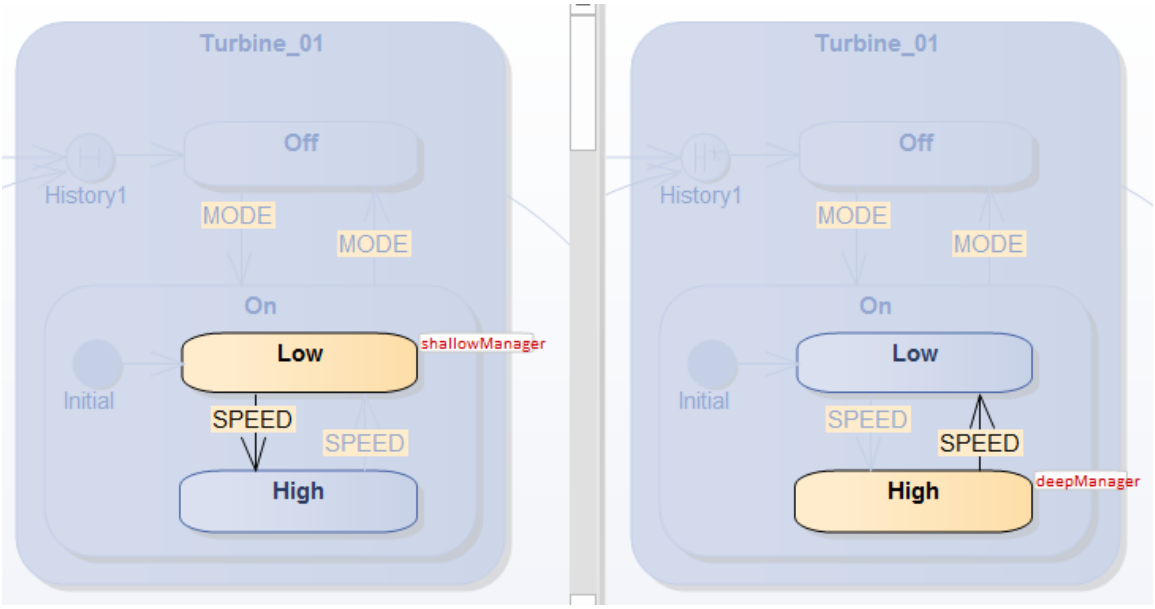
Turbine_02

- Turbine_02.Désactivé

Turbine_03

- Turbine_03.Activée
- Turbine_03.On.Low

Lorsque nous déclenchons davantage Déclencheur NEXT, Turbine_01 sera à nouveau activé.



Cette séquence de traces peut être observée depuis la fenêtre Simulation :

Pour ShallowManager :

Déclencheur [SUIVANT]

01shallowManager[ShallowTurbineManager].StateMachine_Turbine_03_On_Low EXIT

02 ShallowManager[ShallowTurbineManager].StateMachine_Turbine_03_On EXIT

03 ShallowManager[ShallowTurbineManager].StateMachine_Turbine_03 QUITTER

04 ShallowManager[ShallowTurbineManager].Turbine_03__TO__History1_105711_61732 Effet

05 ShallowManager[ShallowTurbineManager].StateMachine_Turbine_01 ENTRÉE

06 ShallowManager[ShallowTurbineManager].StateMachine_Turbine_01 DO

07 ShallowManager[ShallowTurbineManager].StateMachine_Turbine_01_On ENTRY

08shallowManager[ShallowTurbineManager].StateMachine_Turbine_01_On DO

09 ShallowManager[ShallowTurbineManager].Initial_105721__TO__Low_61729 Effet

10 ShallowManager[ShallowTurbineManager].StateMachine_Turbine_01_On_Low ENTRY

11shallowManager[ShallowTurbineManager].StateMachine_Turbine_01_On_Low DO

On peut apprendre que :

- Le nœud ShallowHistory restaure Turbine_01 jusqu'à Turbine_01.On
- Ensuite, la région contenue par Composite State Turbine_01.On sera activée par le nœud *initial* , qui a été activé à *Low.*

Pour deepManager :

Déclencheur [SUIVANT]

01 deepManager[DeepTurbineManager].StateMachine_Turbine_03_On_Low QUITTER

02 deepManager[DeepTurbineManager].StateMachine_Turbine_03_On EXIT

03 deepManager[DeepTurbineManager].StateMachine_Turbine_03 QUITTER

04 deepManager[DeepTurbineManager].Turbine_03__TO__History1_105679_61708 Effet

05 deepManager[DeepTurbineManager].StateMachine_Turbine_01 ENTRÉE

06 deepManager[DeepTurbineManager].StateMachine_Turbine_01 DO

07 deepManager[DeepTurbineManager].StateMachine_Turbine_01_On ENTRY

08 deepManager[DeepTurbineManager].StateMachine_Turbine_01_On_High ENTRY

On peut apprendre que :

- Le nœud *deepHistory* restaure Turbine_01 jusqu'à Turbine_01.On.High

Déclencheur [SUIVANT] pour quitter Turbine_01 et activer Turbine_02

ShallowManager et *deepManager* activent Turbine_02.Off, qui est l'instantané de l'historique à leur sortie.

Déclencheur [SUIVANT] pour quitter Turbine_02 et activer Turbine_03

ShallowManager et *deepManager* activent Turbine_03.On.Low. Cependant, les séquences de *ShallowManager* et *DeepManager* sont différentes.

Pour *ShallowManager* , *ShallowHistory* ne peut être restauré que jusqu'à Turbine_03.On. Puisqu'un nœud *Initial* est défini dans Turbine_03.On, la Transition provenant d' *Initial* sera activée et Turbine_03.On.Low est atteinte.

01shallowManager[ShallowTurbineManager].StateMachine_Turbine_02_Off QUITTER

02 ShallowManager[ShallowTurbineManager].StateMachine_Turbine_02 QUITTER

03 ShallowManager[ShallowTurbineManager].Turbine_02__TO__History3_105713_61725 Effet

04 ShallowManager[ShallowTurbineManager].StateMachine_Turbine_03 ENTRÉE

05shallowManager[ShallowTurbineManager].StateMachine_Turbine_03 DO

06 ShallowManager[ShallowTurbineManager].StateMachine_Turbine_03_On ENTRY

07 ShallowManager[ShallowTurbineManager].StateMachine_Turbine_03_On DO

08 ShallowManager[ShallowTurbineManager].Initial_105727__TO__Low_61728 Effet

09 ShallowManager[ShallowTurbineManager].StateMachine_Turbine_03_On_Low ENTRY

10shallowManager[ShallowTurbineManager].StateMachine_Turbine_03_On_Low DO

Pour *deepManager* , l' *histoire profonde* peut être restaurée directement jusqu'à Turbine_03.On.Low.

01 deepManager[DeepTurbineManager].StateMachine_Turbine_02_Off QUITTER

02 deepManager[DeepTurbineManager].StateMachine_Turbine_02 QUITTER

03 deepManager[DeepTurbineManager].Turbine_02__TO__History3_105680_61701 Effet

04 deepManager[DeepTurbineManager].StateMachine_Turbine_03 ENTRÉE

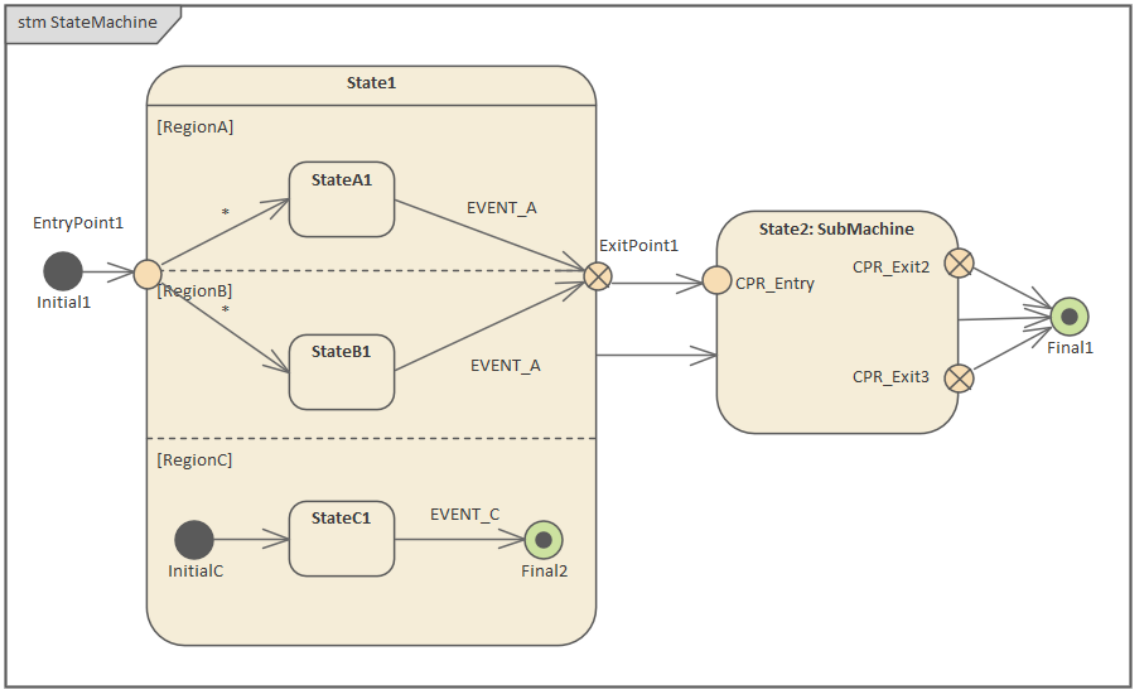
05 deepManager[DeepTurbineManager].StateMachine_Turbine_03 DO

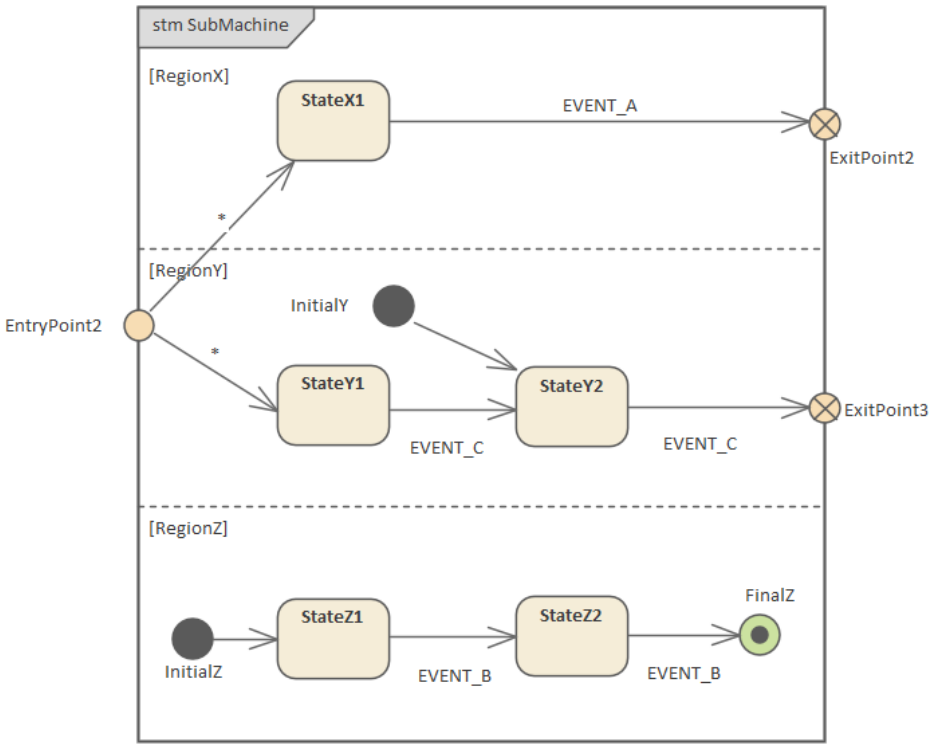
06 deepManager[DeepTurbineManager].StateMachine_Turbine_03_On ENTRY

07 deepManager[DeepTurbineManager].StateMachine_Turbine_03_On_Low ENTRY

Exemple : points d'entrée et de sortie (références de points de connexion)

Enterprise Architect prend en support les points d'entrée et de sortie, ainsi que les références de points de connexion. Dans cet exemple, nous définissons deux Statemachines pour *MyClass : Statemachine* et *SubMachine.*





- *State1* est un State composite (également appelé State orthogonal car il comporte plusieurs régions) avec trois

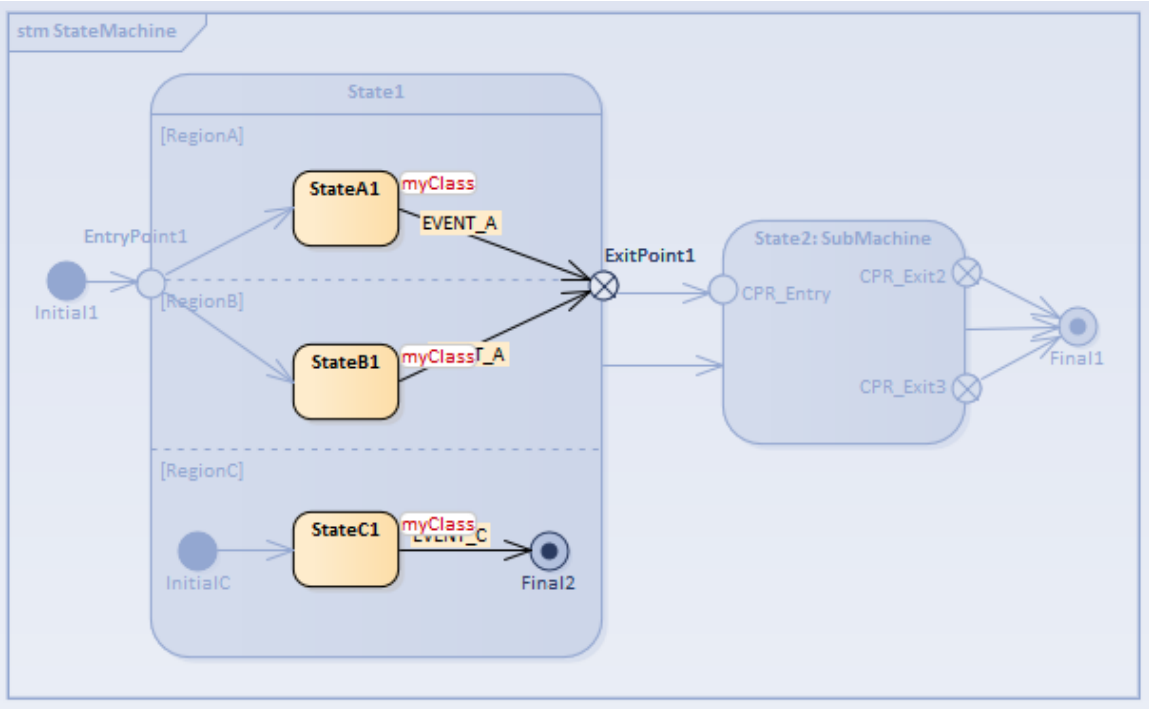
régions : *RegionA* , *RegionB* et *RegionC.*

- *State2* est un State de sous-machine appelant *SubMachine* , qui comporte trois régions : *RegionX* , *RegionY* et *RegionZ*
- *EntryPoint1* est défini sur *State1* pour activer deux des trois régions ; *EntryPoint2* est défini sur *SubMachine* pour activer deux des trois régions
- *ExitPoint1* est défini sur *State1* ; deux points de sortie *ExitPoint2* et *ExitPoint3* sont définis sur *SubMachine*
- Les références de points de connexion sont définies sur *State2* et se lient aux points d'entrée/sortie de la sous-machine de saisie
- Les nœuds initiaux sont définis pour démontrer l'activation par défaut des régions

Entrer d'un State : Entry Point Entry

EntryPoint1* sur *State1

Lorsqu'une transition ciblée sur *EntryPoint1* est activée, *State1* est activé suivi des régions contenues.

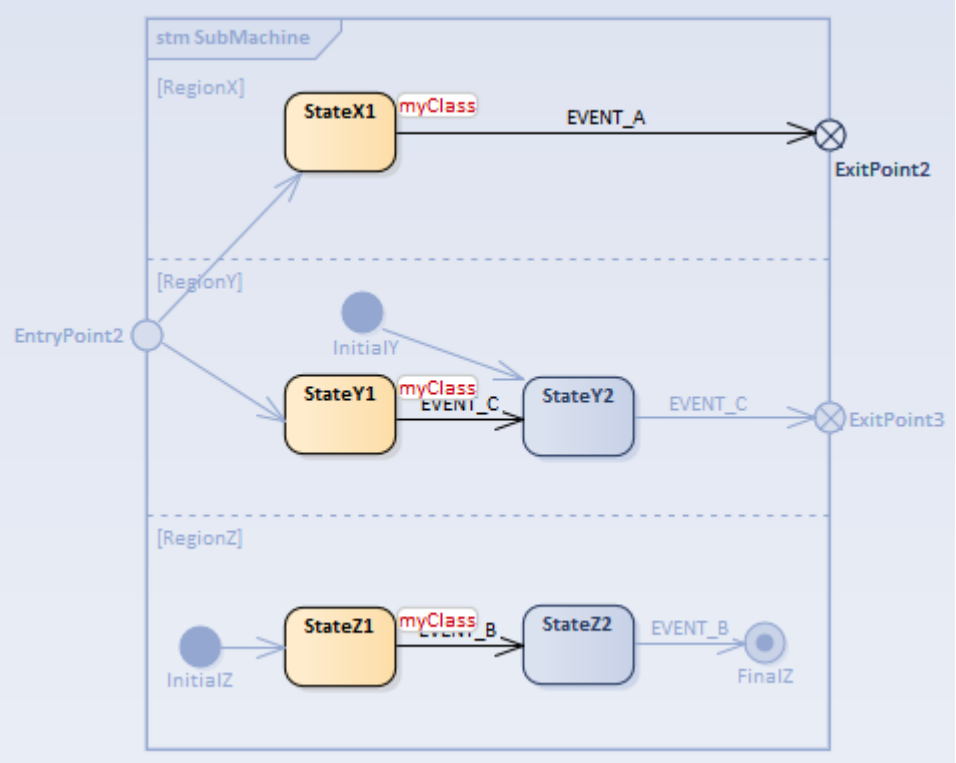


- L'activation explicite se produit pour *RegionA* et *RegionB* , car chacun d'eux est entré par une transition se terminant sur l'un des sommets contenus dans la région.
- L'activation par défaut se produit pour *RegionC* , car elle définit un pseudo-état initial *InitialC* et la transition provenant de *InitialC* vers *StateC1* démarre l'exécution

EntryPoint2* sur *la sous-machine

La Séquence Déclencheur à simuler est : [EVENT_C, EVENT_A].

Lorsqu'une transition ciblée sur la référence de point de connexion *CPR_Entry* sur *State2* est activée, *State2* est activé, suivi de l'activation de la sous-machine via les points d'entrée de liaison.



- L'activation explicite se produit pour *RegionX* et *RegionY* , car chacun d'eux est entré par une transition se terminant sur l'un des sommets contenus dans la région - *StateX1* dans *RegionX* , *StateY1* dans *RegionY*
- L'activation par défaut se produit pour *RegionZ* , car elle définit un pseudo-état initial *InitialZ* et la transition provenant de *InitialZ* vers *StateZ1* démarre l'exécution

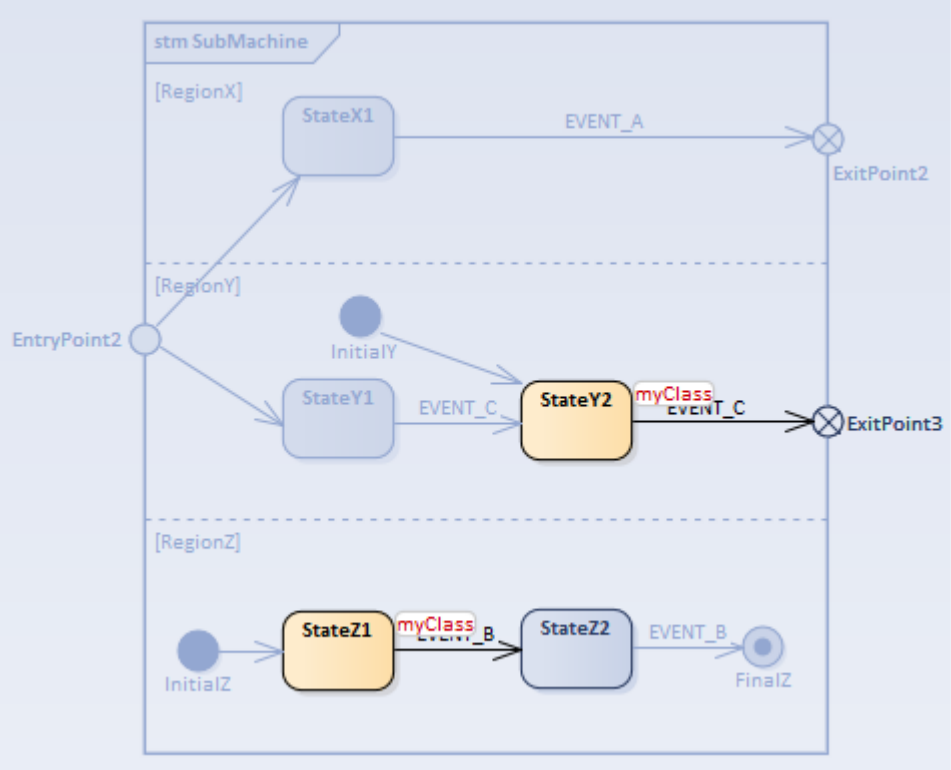
Entrer d'un State : Entrée par défaut

Cette situation se présente lorsque l' State composite est la cible directe d'une transition.

Entrée par défaut de *l'État 2*

La Séquence Déclencheur à simuler est : [EVENT_A, EVENTC].

Lorsqu'une transition ciblée directement sur *State2* est activée, *State2* est activé, suivi d'une activation par défaut pour toutes les régions de la sous-machine.



- State *de RegionX* est inactif car il ne définit pas de nœud initial
- *RegionY* est activé via *InitialY* et la transition vers *StateY2* est exécutée
- *RegionZ* est activé via *InitialZ* et la transition vers *StateZ1* est exécutée

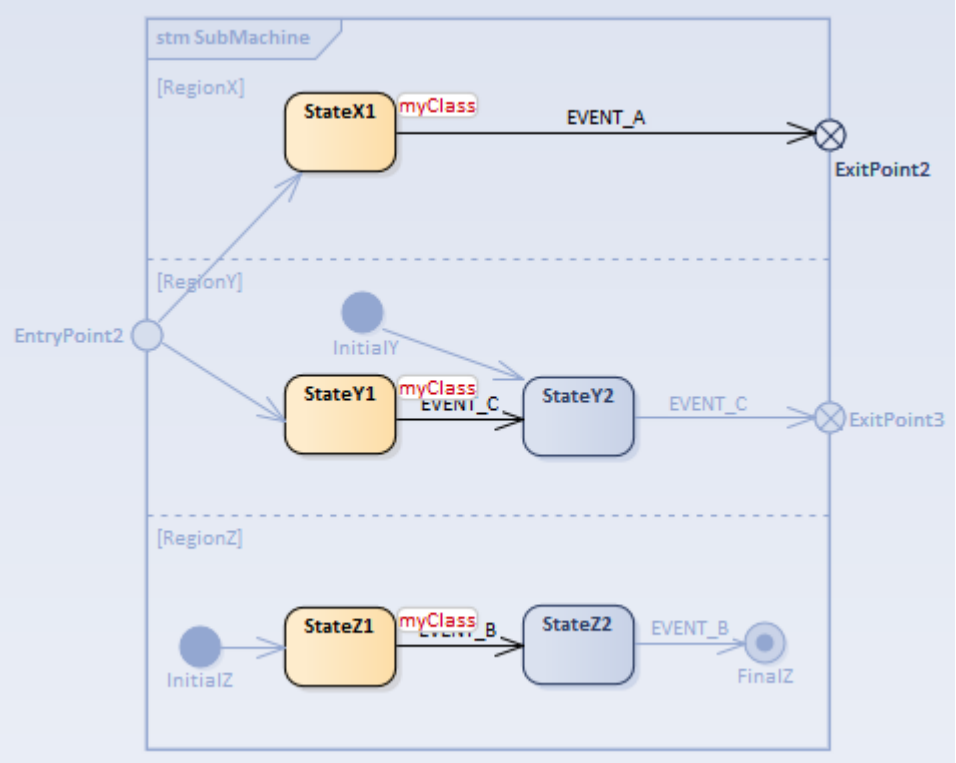
Sortie State

État1 Quitter

- Séquence Déclencheur [EVENT_C, EVENT_A] : *RegionC* est inactivée en premier, puis *RegionA* et *RegionB* ; une fois le comportement de sortie de *State1* exécuté, la transition sortant de *ExitPoint1* est activée
- Déclencheur Séquence [EVENT_A, EVENT_C] : *RegionA* et *RegionB* sont inactivées en premier, puis *RegionC* ; une fois le comportement de sortie de *State1* exécuté, la transition sortant directement de *State1* est activée

État2 Sortie

Déclencheur Séquence [EVENT_C, EVENT_A], donc l'état actuel ressemble à ceci :



- Déclencheur Séquence [EVENT_A, EVENT_C, EVENT_C, EVENT_B, EVENT_B] : *RegionX* est inactivé en premier, puis *RegionY* , et *RegionZ* est le dernier ; une fois le comportement de sortie de *State2* exécuté, la transition sortant directement de *State2* est activée
- Déclencheur Séquence [EVENT_A, EVENT_B, EVENT_B, EVENT_C, EVENT_C] : *RegionX* est inactivé en premier, puis *RegionZ* , et *RegionY* est le dernier ; une fois le comportement de sortie de *State2* exécuté, la transition sortant de *CPR_Exit3* est activée (*ExitPoint3* sur *SubMachine* est lié à *CPR_Exit3* de *State2*)
- Déclencheur Séquence [EVENT_C, EVENT_C, EVENT_B, EVENT_B, EVENT_A] : *RegionY* est inactivé en premier, puis *RegionZ* , et *RegionX* est le dernier ; une fois le comportement de sortie de *State2* exécuté, la transition sortant de *CPR_Exit2* est activée (*ExitPoint2* sur *SubMachine* est lié à *CPR_Exit2* de *State2*)

Exemple : Motif d'événement différé

Enterprise Architect supporte le Motif d'événement différé.

Pour créer un événement différé dans un State :

1. Créer une auto transition pour l' State .
2. Changez le « genre » de la transition en « interne ».
3. Spécifiez le Déclencheur comme étant l'événement que vous souhaitez différer.
4. Dans le champ « Effet », tapez « defer(); ».

Simuler:

1. Sélectionnez « Simuler > Simulation Dynamique > Simulateur > Ouvrir la fenêtre Simulation ». Sélectionnez également 'Simuler > Simulation Dynamique > Événements ' pour ouvrir la fenêtre Simulation Événements .
2. La fenêtre Événements du Simulateur vous aide à déclencheur des événements ; double-cliquez sur un déclencheur dans la colonne ' Déclencheurs en Attente '.
3. La fenêtre Simulation montre l'exécution en texte. Vous pouvez taper « dump » dans la ligne de commande du simulateur pour afficher le nombre d'événements différés dans la file d'attente ; le résultat pourrait ressembler à ceci : 24850060] Pool d'événements : [NOUVEAU, NOUVEAU, NOUVEAU, NOUVEAU, NOUVEAU,]

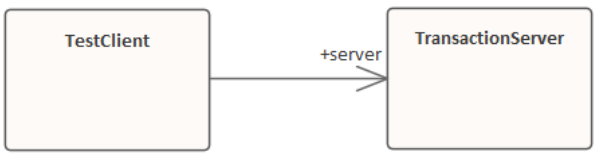
Exemple d'événement différé

Cet exemple montre un modèle utilisant Événements différés et la fenêtre Simulation Événements affichant tous Événements disponibles.

Nous définissons d'abord les contextes (les éléments Class contenant les Statemachines), les simulons dans un contexte simple et déclenchons l'événement depuis l'extérieur de celui-ci ; puis simulez dans un contexte client-serveur avec le mécanisme d'événement Send.

Créer un contexte et Statemachine

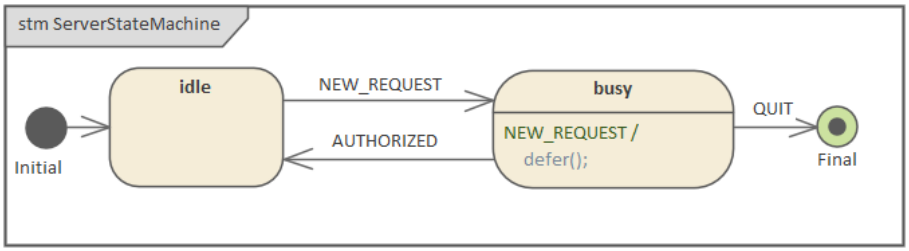
Créer le contexte du serveur



Créez un diagramme de classes et :

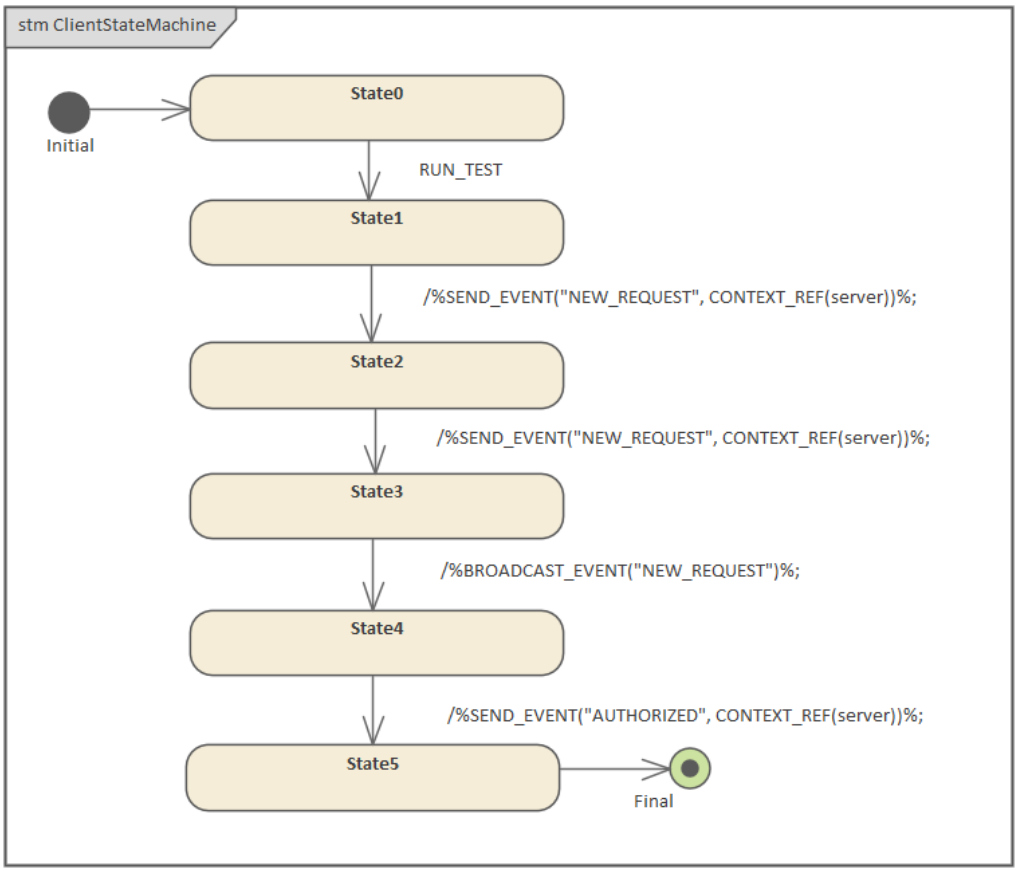
1. Un élément de classe *TransactionServer* , auquel vous ajoutez un Statemachine *ServerStateMachine* .
2. Un élément de classe *TestClient* , auquel vous ajoutez un Statemachine *ClientStateMachine* .
3. Une association de *TestClient* à *TransactionServer* , avec le rôle cible nommé *serveur.*

Modélisation pour *ServerStateMachine*



1. Ajoutez un nœud initial *initial* au diagramme Statemachine et passez à un State *inactif* .
2. Transition (avec événement NEW_REQUEST comme Déclencheur) vers un State *occupé* .
3. Transition (avec événement QUIT comme Déclencheur) vers un State Final *Final.*
4. Transition (avec événement AUTORISÉ comme Déclencheur) au *ralenti* .
5. Transition (avec l'événement NEW_REQUEST comme Déclencheur et *defer();* comme effet) vers *occupé*

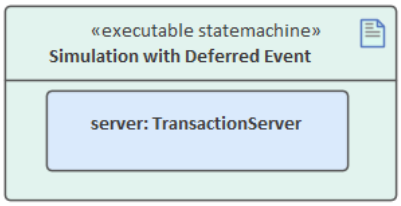
Modélisation pour *ClientStateMachine*



1. Ajoutez un nœud initial *initial* au diagramme Statemachine et passez à un State *State0* .
2. Transition (avec événement RUN_TEST comme déclencheur) vers un State *State1* .
3. Transition (avec effet : %SEND_EVENT("NEW_REQUEST", CONTEXT_REF(server))%;) vers un State *State2.*
4. Transition (avec effet : %SEND_EVENT("NEW_REQUEST", CONTEXT_REF(server))%;) vers un State *State3* .
5. Transition (avec effet : %BROADCAST_EVENT("NEW_REQUEST")%;) vers un State *State4* .
6. Transition (avec effet : %SEND_EVENT("AUTHORIZED", CONTEXT_REF(server))%;) vers un State *State5* .
7. Transition vers un State final *final.*

Simulation dans un contexte simple

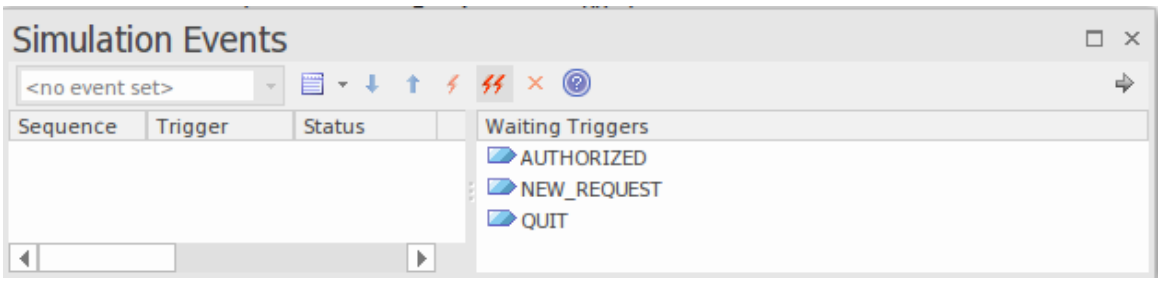
Créer l'artefact Simulation



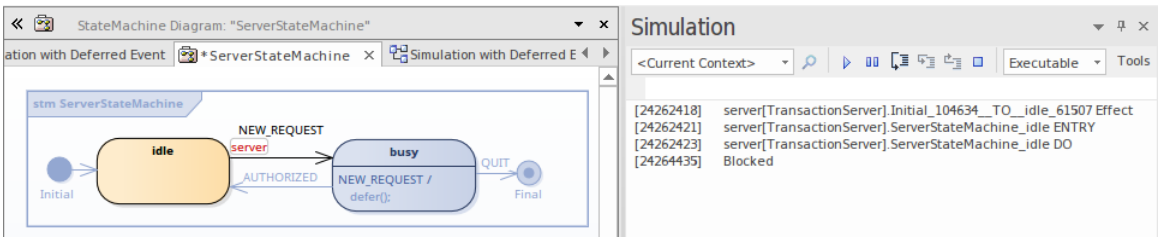
1. Créez un artefact Statemachine Exécutable avec le nom *Simulation with Deferred Event* et le champ 'Language' défini sur *JavaScript* .
2. Agrandissez-le, puis Ctrl+faites glisser l'élément *TransactionServer* sur l'artefact et collez-le en tant que propriété avec le *serveur de noms.*

Exécuter la Simulation

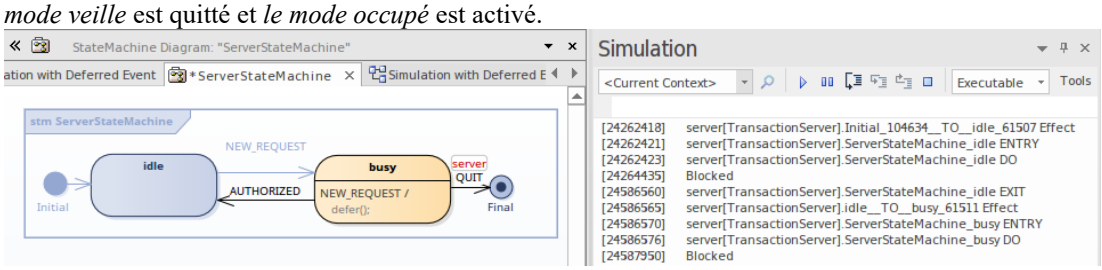
1. Sélectionnez l'Artefact, puis sélectionnez l'option 'Simulate > États Exécutables > Statemachine > Générer , Build and Exécuter ', et spécifiez un répertoire pour votre code (Note : tous les fichiers du répertoire seront supprimés avant le démarrage de la simulation).
2. Cliquez sur le bouton Générer .
3. Sélectionnez l'option 'Simuler > Simulation Dynamique > Événements ' pour ouvrir la fenêtre Événement Simulation .



Lorsque la simulation démarre, *l'inactivité* sera l'état actif.

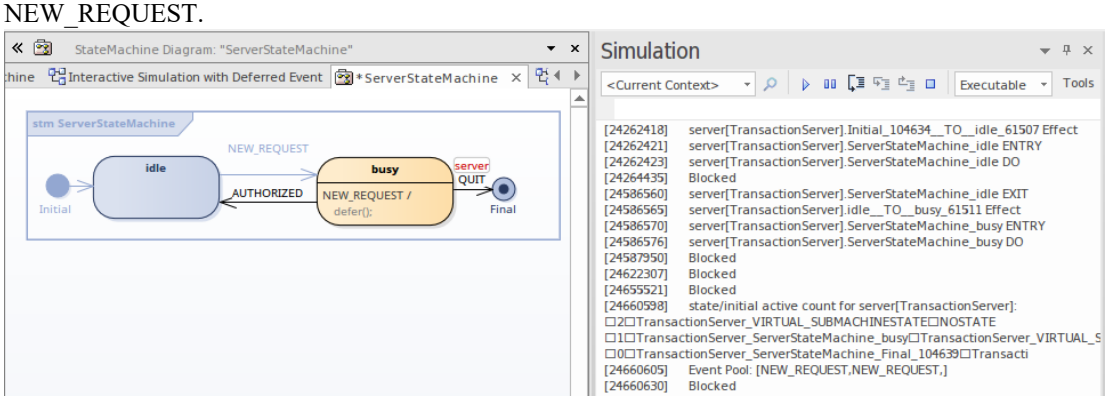


1. Double-cliquez sur NEW_REQUEST dans la fenêtre Simulation Event pour l'exécuter en tant que Déclencheur ; *le mode veille* est quitté et *le mode occupé* est activé.

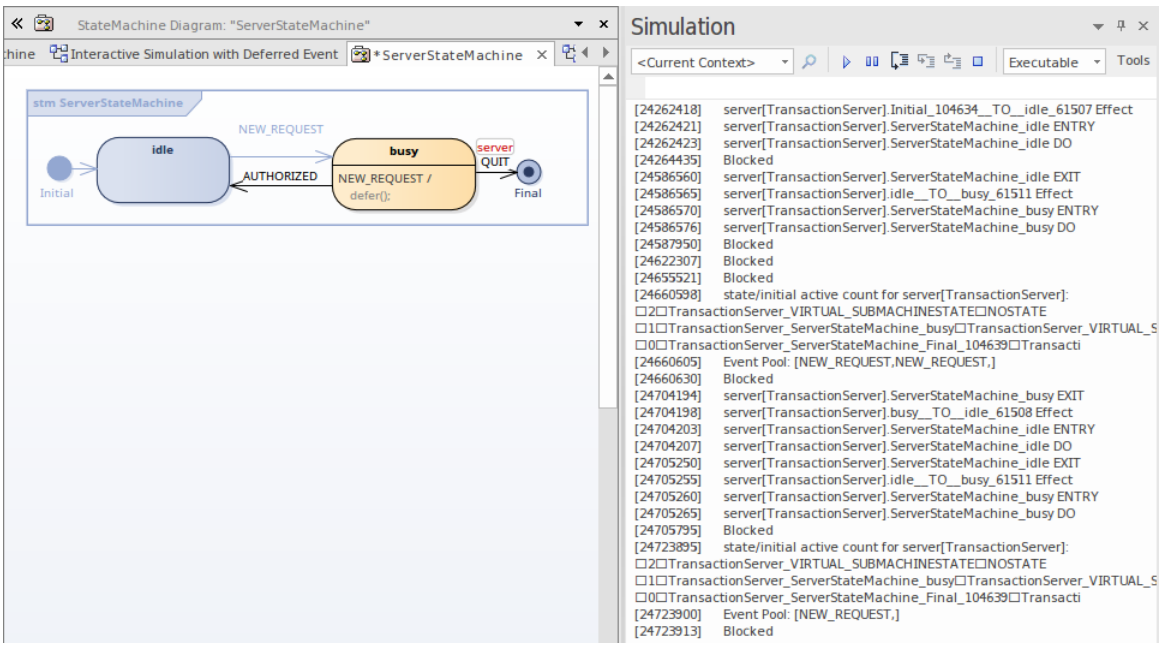


2. Double-cliquez sur NEW_REQUEST dans la fenêtre Simulation Event pour l'exécuter à nouveau en tant que Déclencheur ; *occupé* reste activé et une instance de NEW_REQUEST est ajoutée au pool d'événements.

3. Double-cliquez sur NEW_REQUEST dans la fenêtre Simulation Event pour l'exécuter une troisième fois en tant que Déclencheur ; *occupé* reste activé et une instance de NEW_REQUEST est ajoutée au pool d'événements.
4. Type *dump* dans la ligne de commande de la fenêtre Simulation ; notez que le pool d'événements a deux instances de NEW_REQUEST.

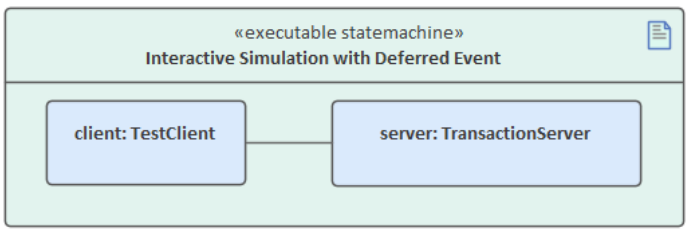


5. Double-cliquez sur AUTORISÉ dans la fenêtre Simulation Event pour l'exécuter en tant que Déclencheur ; ces actions ont lieu :
   - *occupé* est quitté et *inactif* devient actif
   - un événement NEW_REQUEST est récupéré du pool, *inactif* est quitté et *occupé* devient actif
6. Type *dump* dans la ligne de commande de la fenêtre Simulation ; il n'y a maintenant qu'une seule instance de NEW_REQUEST dans le pool d'événements.



Simulation interactive via Envoyer/Diffuser un événement

Créer l'artefact Simulation

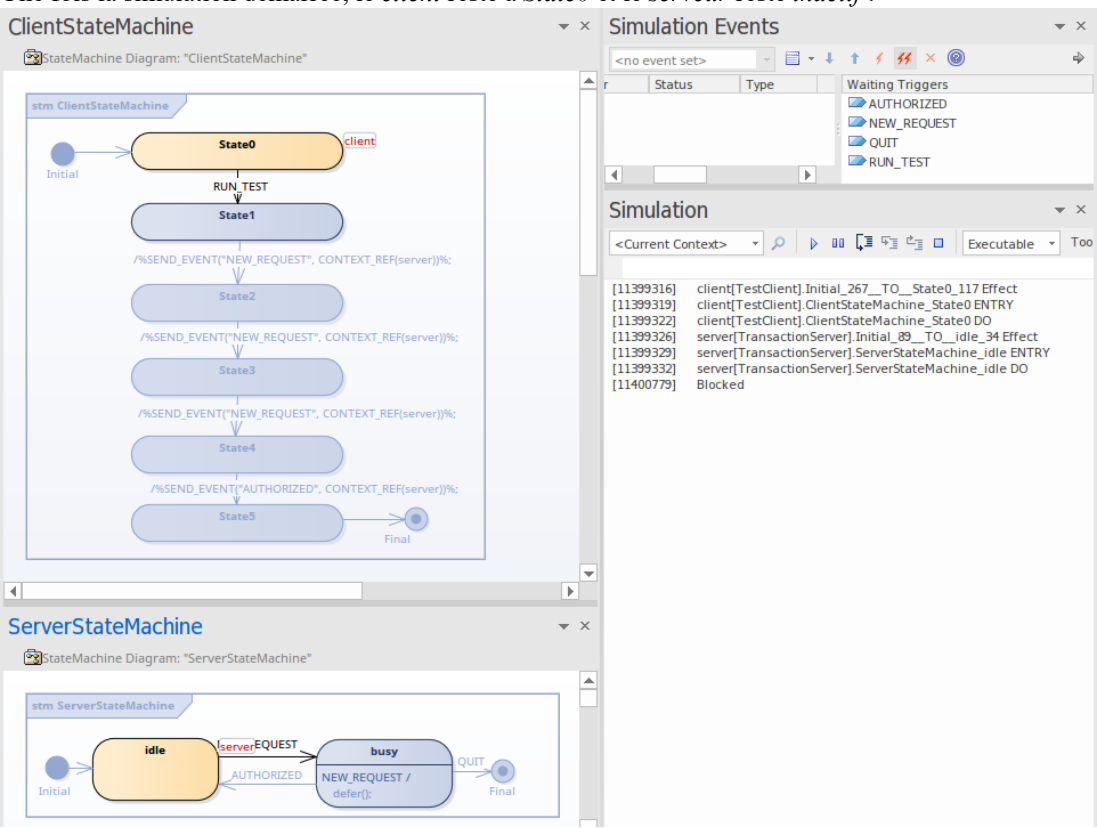


1. Créer un artefact Statemachine Exécutable avec le nom *Interactive Simulation with Deferred Event* et le champ 'Language' défini sur *JavaScript ;* agrandir l'élément.
2. Ctrl+Faites glisser l'élément *TransactionServer* sur l'artefact et collez-le en tant que propriété avec le *serveur de noms.*
3. Ctrl+Faites glisser l'élément *TestClient* sur l'artefact et collez-le en tant que propriété avec le nom *client* .
4. Créez un connecteur du *client* au *serveur* .
5. Cliquez sur le connecteur et appuyez sur Ctrl+L pour sélectionner l'association de l'élément *TestClient* à l'élément *TransactionServer* .

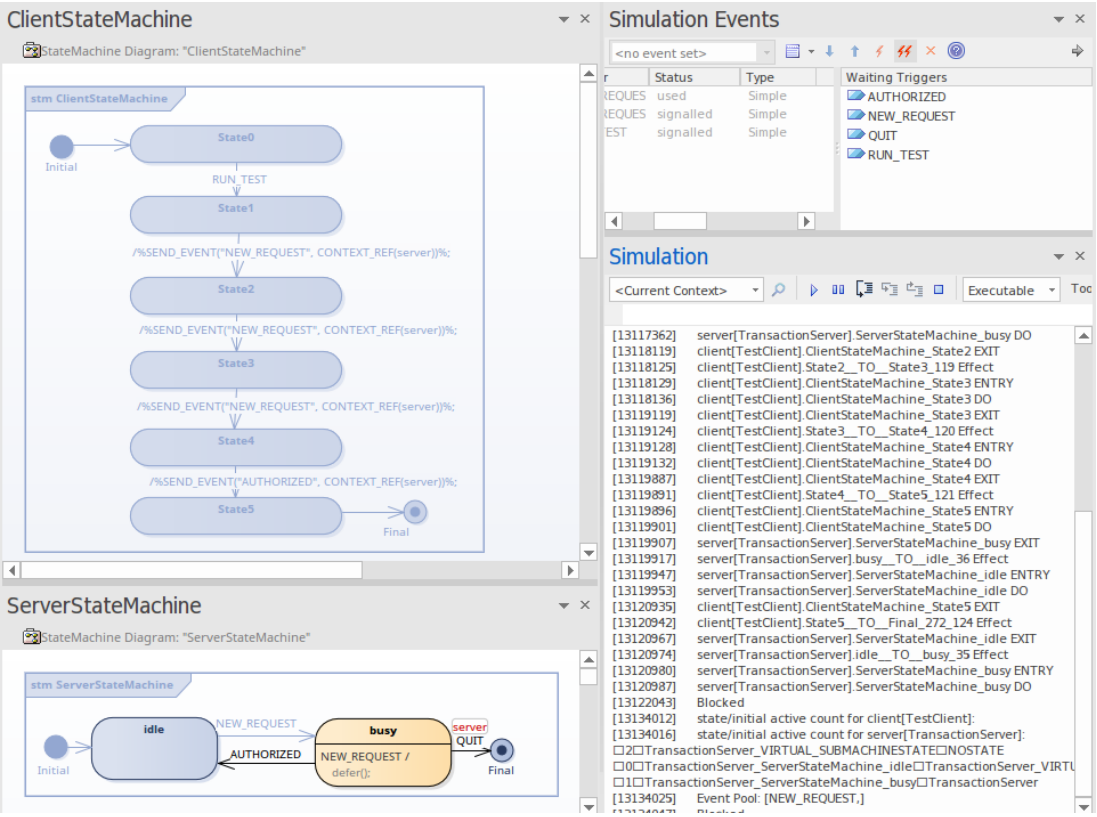
Exécuter Simulation interactive

1. Lancez la simulation de la même manière que pour le contexte simple.

   Une fois la simulation démarrée, le *client* reste à *State0* et le *serveur* reste *inactif* .



2. Double-cliquez sur RUN_TEST dans la fenêtre Simulation Event pour le déclencheur . L'événement NEW_REQUEST sera déclenché trois fois (par SEND_EVENT et BROADCAST_EVENT) et AUTHORIZED sera déclenché une fois par SEND_EVENT.



Type *dump* dans la ligne de commande de la fenêtre Simulation . Il reste une instance de NEW_REQUEST dans le pool d'événements. Le résultat correspond à notre test de déclenchement manuel.

Débogage de l'exécution des Statemachines Exécutables

La création de Statemachines Exécutables offre des avantages même après la génération du code. Grâce à l' Analyseur d'Exécution , Enterprise Architect est capable de se connecter au code généré. En conséquence, vous êtes en mesure de déboguer visuellement et de vérifier le comportement correct du code ; exactement le même code généré à partir de vos Statemachines , démontré par la simulation et finalement incorporé dans un système du monde réel.

Débogage d'une Statemachine

Être capable de déboguer une Statemachine Exécutable offre des avantages supplémentaires, tels que la possibilité de :

- Interrompre l'exécution de la simulation et de toutes Statemachines en cours d'exécution
- Vue l'état brut de chaque instance Statemachine impliquée dans la simulation
- Vue le code source et Pile d'Appel à tout moment
- Tracez des informations supplémentaires sur l'état d'exécution grâce au placement de points de trace sur les lignes du code source
- Contrôler l'exécution grâce à l'utilisation de points d'action et de points d'arrêt (arrêt sur erreur par exemple)
- Diagnostiquer les changements de comportement, dus à des changements de code ou modélisation

Si vous avez généré, construit et exécuter un Statemachine Exécutable avec succès, vous pouvez le déboguer ! Le script Analyzer créé pendant le processus de génération est déjà configuré pour fournir le débogage. Pour démarrer le débogage, commencez simplement à exécuter Statemachine Exécutable à l'aide du contrôle Simulation . Cependant, en fonction de la nature du comportement débogué, nous définirions probablement d'abord certains points d'arrêt.

Interrompre l'exécution lors d'une transition d'état

Comme n'importe quel débogueur, nous pouvons utiliser des points d'arrêt pour examiner la Statemachine en cours d'exécution à un moment donné du code. Localisez une classe qui vous intéresse dans la fenêtre du diagramme ou Navigateur et appuyez sur F12 pour afficher le code source. Il est facile de localiser le code des transitions State à partir des conventions de dénomination utilisées lors de la génération. Si vous souhaitez interrompre une transition particulière, localisez la fonction de transition dans l'éditeur et placez un marqueur de point d'arrêt en cliquant dans la marge gauche sur une ligne de la fonction. Lorsque vous exécuter le Statemachine Exécutable , le débogueur s'arrêtera à cette transition et vous pourrez visualiser l'état brut des variables de toutes Statemachines impliquées.

Interrompre l'exécution sous condition

Chaque point d'arrêt peut prendre une condition et une instruction de trace. Lorsque le point d'arrêt est rencontré et que la condition est évaluée à True, l'exécution s'arrête. Sinon, l'exécution se poursuivra normalement. Vous composez la condition en utilisant les noms des variables brutes et en les comparant à l'aide des opérandes d'égalité standard : < > = >= <=. Par exemple:

(this.m_nCount > 100) et (this.m_ntype == 1)

Pour ajouter une condition à un point d'arrêt que vous avez défini, cliquez-droit sur le point d'arrêt et sélectionnez ' Propriétés '. En cliquant sur le point d'arrêt tout en appuyant sur la touche Ctrl, les propriétés peuvent être rapidement modifiées.

Information auxiliaire Traçage

Il est possible de tracer des informations à partir de la Statemachine elle-même en utilisant la clause TRACE dans, par exemple, un *effect* . Le débogage fournit également fonctionnalités de trace appelées Tracepoints. Ce sont simplement des points d'arrêt qui, au lieu de se rompre, impriment des instructions de trace lorsqu'elles sont rencontrées. La sortie est affichée dans la fenêtre Simulation Control. Ils peuvent être utilisés comme aide au diagnostic pour montrer et prouver la séquence des événements et l'ordre dans lequel les instances changent d'état.

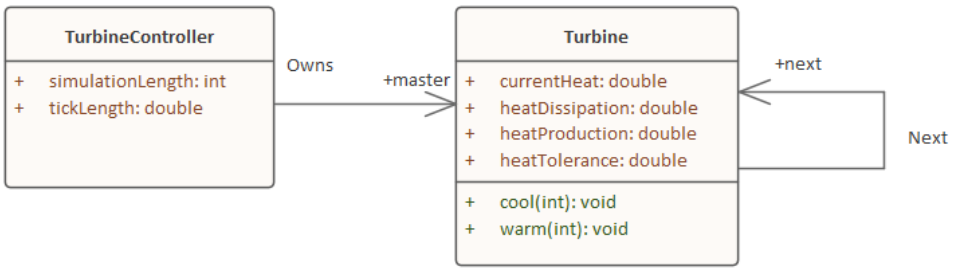
Visualisation de la Pile d'Appel

Chaque fois qu'un point d'arrêt est rencontré, la Pile d'Appel est disponible dans le menu Analyseur. Utilisez-le pour déterminer l'ordre dans lequel l'exécution a lieu.

Exemple : Statemachine Exécutable

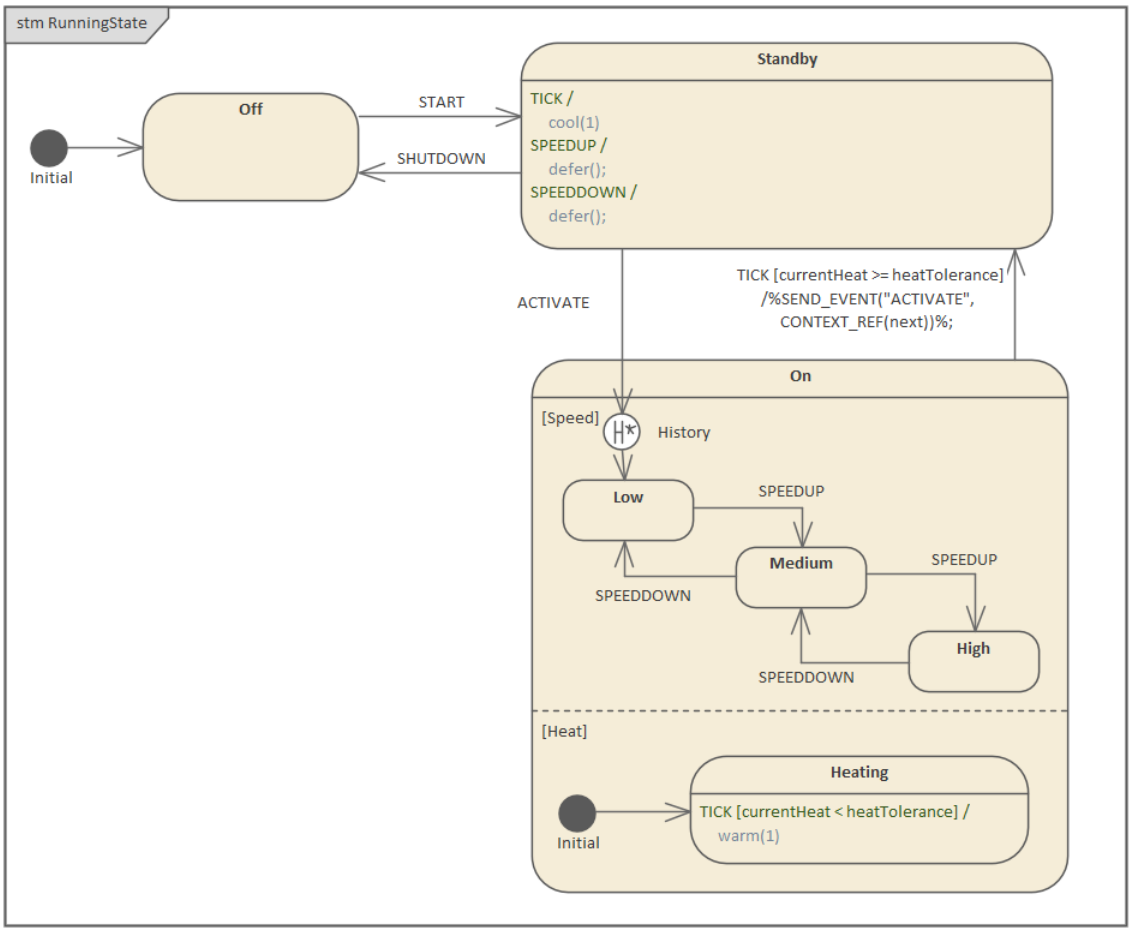
Exemple Modèle de classe

Cette image montre un exemple de modèle de classe utilisé par les Statemachines décrites dans cette rubrique.

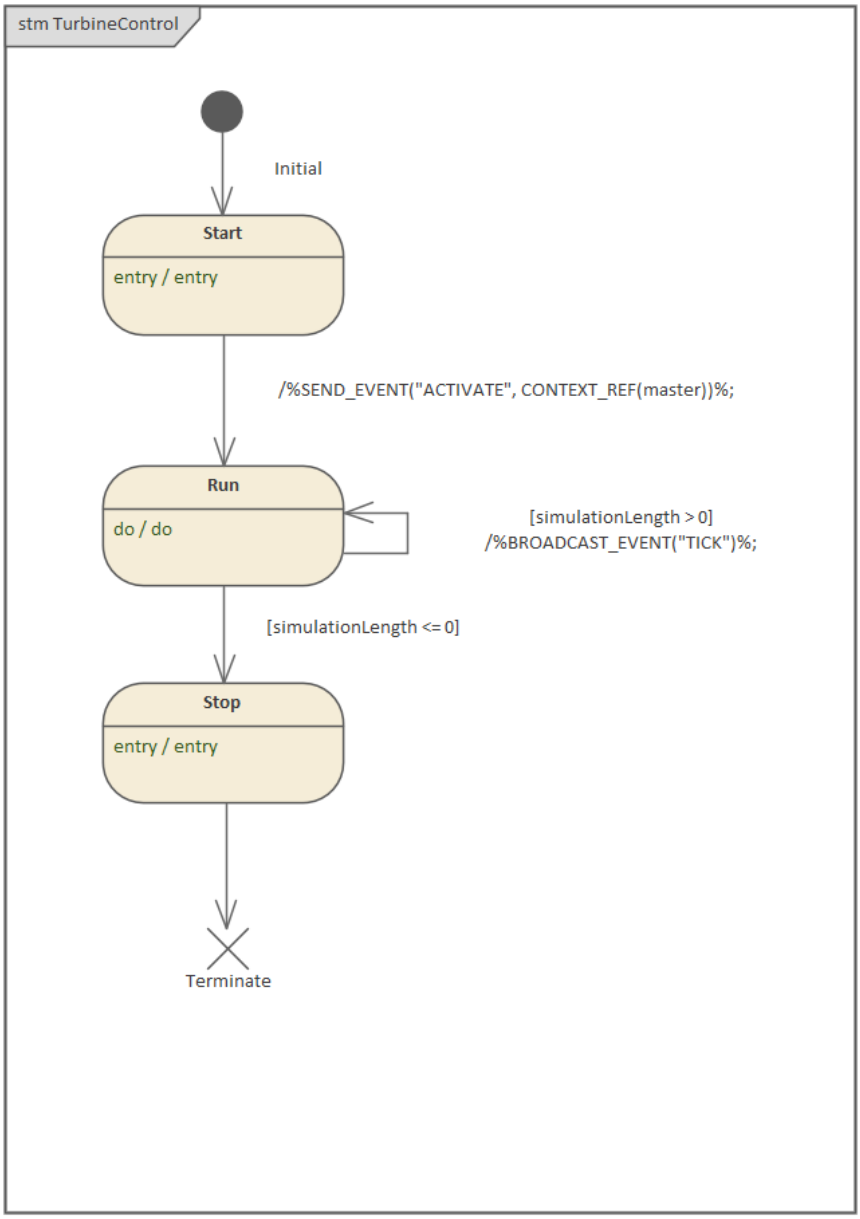


Exemples Statemachines

Ces deux diagrammes montrent les définitions de deux Statemachines . Le premier fait référence à une autre Statemachine du même type, tandis que le second pilote toutes les instances du premier qui existent.

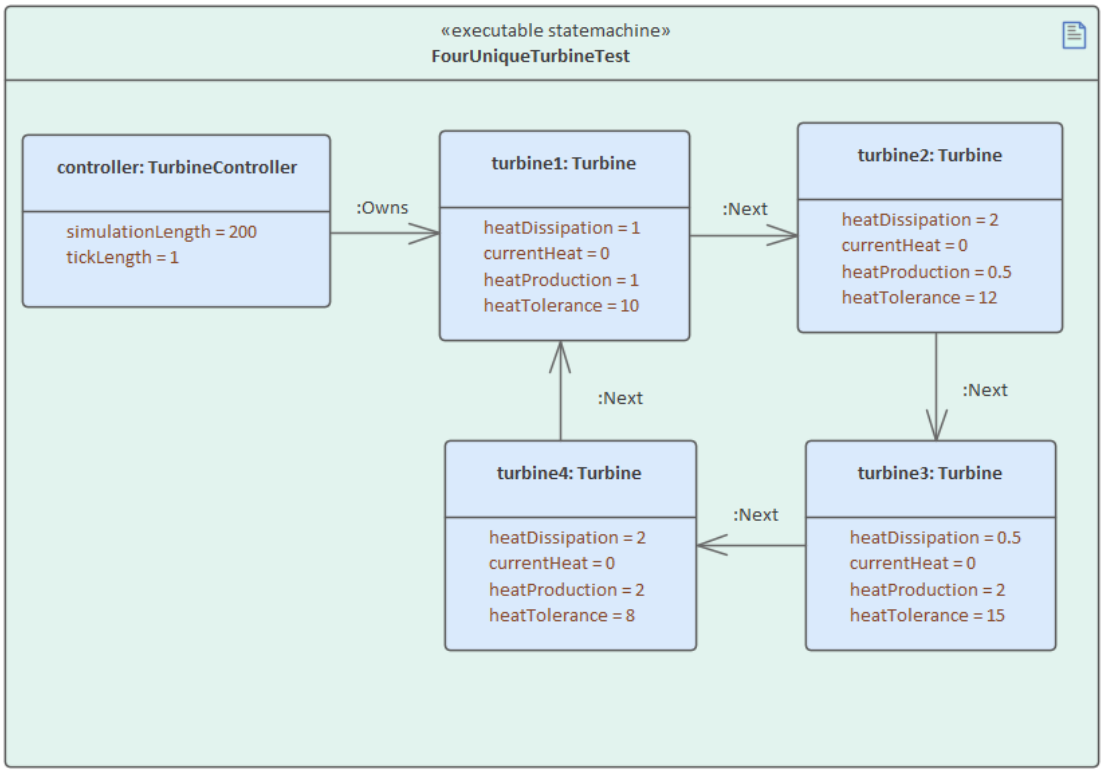


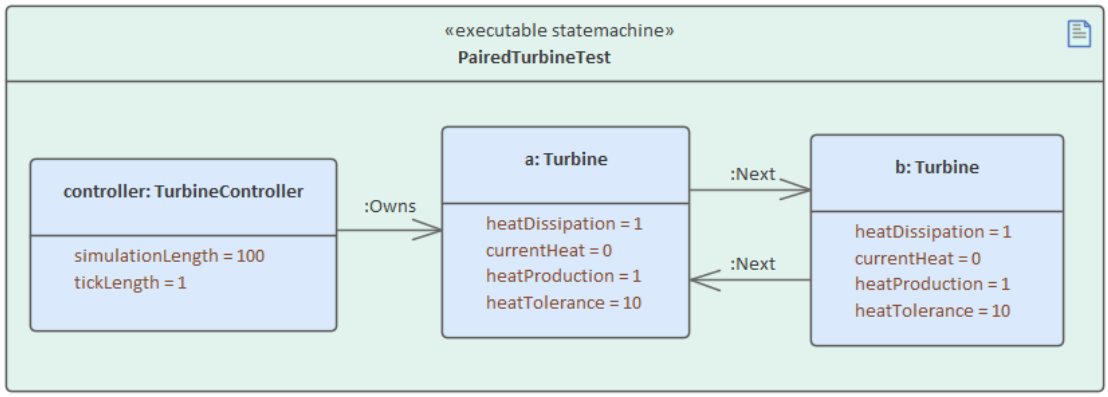
Le contrôleur de niveau supérieur.



Exemples d'artefacts

À partir des exemples diagrammes de classes et Statemachine , nous pouvons créer Statemachines Exécutables comme indiqué ici.

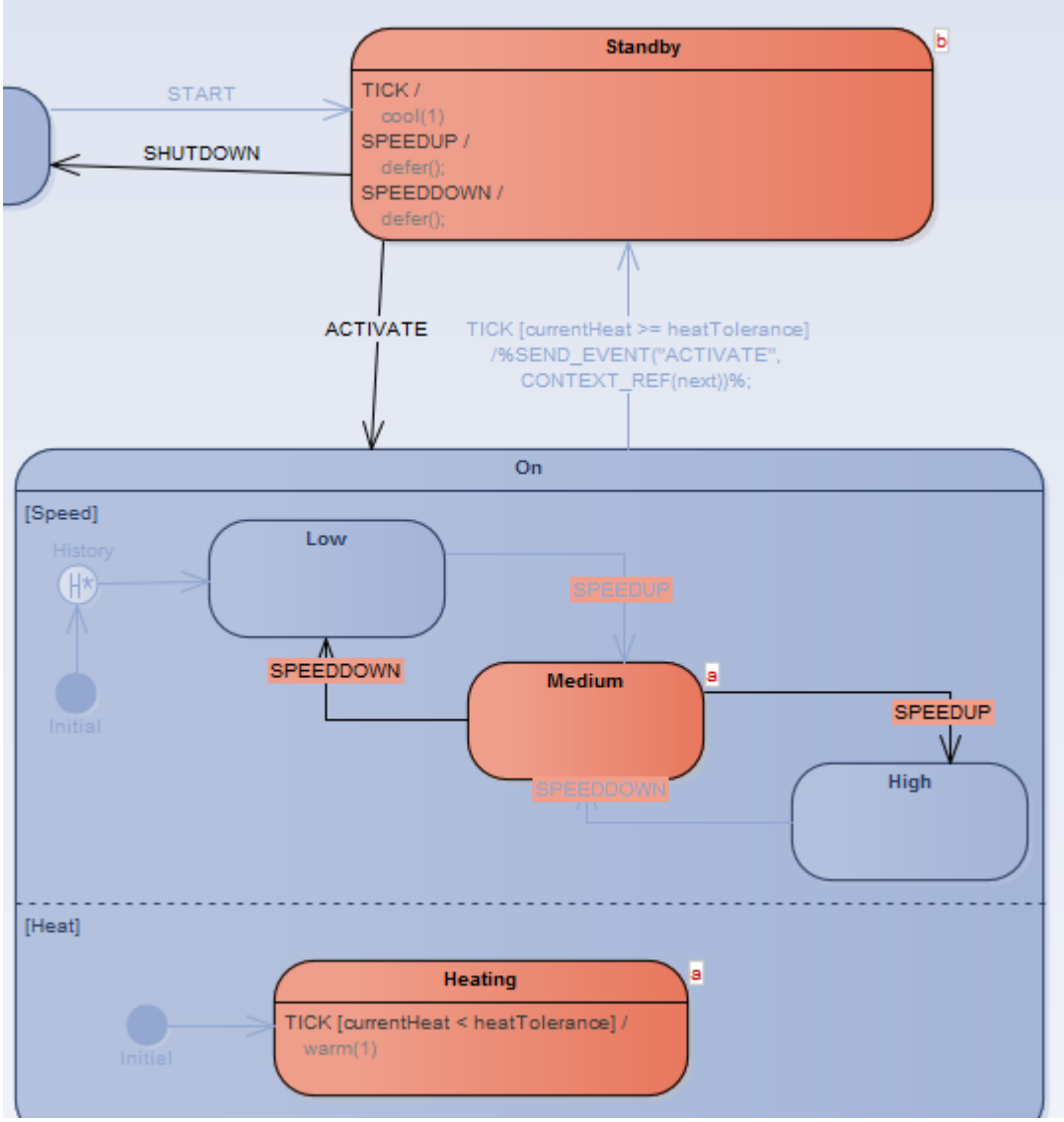




Note comment les valeurs de propriété ont été définies pour chaque propriété et les liens entre les éléments identifient les relations qui existent dans le modèle de classe.

Résultats Simulation

Lors de l'exécution d'une simulation, Enterprise Architect mettra en évidence les States actuellement actifs dans toutes Statemachines . Lorsqu'il existe plusieurs instances d'un Statemachine , il affichera également les noms de chaque instance dans cet State .


Standby
b
START
SHUTDOWN
TICK /
cool(1)
SPEEDUP /
defer();
SPEEDDOWN /
defer();
ACTIVATE
TICK [currentHeat >= heatTolerance]
/%SEND_EVENT("ACTIVATE",
CONTEXT_REF(next))%;
On
[Speed]
History
Low
SPEEDUP
SPEEDDOWN
Medium
a
SPEEDUP
High
SPEEDDOWN
Initial
[Heat]
Heating
a
TICK [currentHeat < heatTolerance] /
warm(1)
Initial

Artefact Statemachine Exécutable

Un artefact Statemachine Exécutable est essentiel pour générer Statemachines qui peuvent interagir les unes avec les autres. Il précise les objets qui seront impliqués dans une simulation, leur état et comment ils se connectent. Un gros avantage de l'utilisation Statemachine Exécutable Artifacts est que chacune des plusieurs parties d'un Artifact peut représenter une instance d'un Statemachine , vous pouvez donc configurer des simulations en utilisant plusieurs instances de chaque Statemachine et observer comment elles interagissent. Un exemple est fourni dans la rubrique d'aide *Exemple Statemachine Exécutable* .

Création des Propriétés d'une Statemachine Exécutable

Chaque scénario Statemachine Exécutable implique une ou plusieurs Statemachines . Les Statemachines incluses sont spécifiées par les éléments de propriété UML ; chaque propriété aura un classificateur UML (classe) qui détermine la ou les Statemachine incluses pour ce type. Plusieurs types inclus en tant que Propriétés multiples peuvent finir par inclure de nombreux Statemachines , qui sont tous créés dans le code et initialisés lors de l'exécution.

Action	Description
Déposez une classe de la fenêtre du Navigateur sur l'artefact « Statemachine Exécutable »	Le moyen le plus simple de définir des propriétés sur une Statemachine Exécutable est de déposer la classe sur la Statemachine Exécutable à partir de la fenêtre Navigateur . Dans le dialogue qui s'affiche, sélectionnez l'option permettant de créer une propriété. Vous pouvez ensuite spécifier un nom décrivant comment le Statemachine Exécutable fera référence à cette propriété. Note : Selon vos options, vous devrez peut-être maintenir la touche Ctrl enfoncée pour choisir de créer une propriété. Ce comportement peut être modifié à tout moment en utilisant la case à cocher 'Maintenir Ctrl pour afficher cette boîte dialogue '.
Utiliser et connecter plusieurs Propriétés UML	Un Statemachine Exécutable décrit l'interaction de plusieurs Statemachines . Il peut s'agir de différentes instances du même Statemachine , Statemachines différentes pour la même instance ou Statemachines complètement différentes provenant de différents types de base. Pour créer plusieurs propriétés qui utiliseront le même Statemachine , déposez plusieurs fois la même classe sur l'artefact. Pour utiliser différents types, supprimez différentes classes de la fenêtre Navigateur selon vos besoins.

Définition de l' State initial des Propriétés

Les Statemachines exécuter par un Statemachine Exécutable seront toutes exécuter dans le contexte de leur propre instance de Class. Un Statemachine Exécutable vous permet de définir l'état initial de chaque instance en attribuant des valeurs de propriété à divers attributs de classe. Par exemple, vous pouvez spécifier l'âge, la taille, le poids ou similaire d'un joueur si ces propriétés sont pertinentes pour le scénario en cours exécuter . Ce faisant, il est possible de définir des conditions initiales détaillées qui influenceront le déroulement du scénario.

Action	Description
Boîte Dialogue Définir les valeurs des propriétés	La dialogue d'attribution des valeurs de propriété peut être ouverte en cliquant avec le bouton droit sur une propriété et en sélectionnant ' Fonctionnalités | Définir les valeurs de propriété', ou en utilisant le raccourci clavier Ctrl+Maj+R.
Attribuer une valeur	La dialogue « Définir les valeurs de propriété » vous permet de définir des valeurs pour tout attribut défini dans la classe d'origine. Pour ce faire, sélectionnez la

	variable, définissez l'opérateur sur '=' et entrez la valeur requise.

Définir Relations entre Propriétés

En plus de décrire les valeurs à attribuer aux variables appartenant à chaque propriété, un Statemachine Exécutable vous permet de définir comment chaque propriété peut en référencer d'autres en fonction du modèle de classe dont elle est une instance.

Action	Description
Créer un connecteur	Connectez plusieurs propriétés à l'aide de la relation Connecteur de la boîte à outils Composite. Vous pouvez également utiliser Quick Linker pour créer une relation entre deux Propriétés et sélectionner « Connecteur » comme type de relation.
Mapper vers la classe Modèle	Une fois qu'un connecteur existe entre deux propriétés, vous pouvez le mapper à l'association qu'il représente dans le modèle de classe. Pour cela, sélectionnez le connecteur et utilisez le raccourci clavier Ctrl+L. La dialogue « Choisir une association » s'affiche, ce qui permet à la Statemachine générée d'envoyer des signaux à l'instance remplissant le rôle spécifié dans la relation lors de l'exécution.

Exécution et Simulation de Statemachines Exécutables

L'une des nombreuses fonctionnalités d' Enterprise Architect est sa capacité à effectuer des simulations. Un Statemachine Exécutable généré et intégré dans Enterprise Architect peut se connecter à Simulation facilités pour démontrer visuellement l'exécution en direct de l'artefact Statemachine .

Démarrer une Simulation

La barre d'outils Contrôle Simulation fournit un bouton Rechercher que vous utilisez pour sélectionner l'artefact Statemachine Exécutable à exécuter . Le contrôle maintient une liste déroulante des Statemachines Exécutables les plus récentes parmi lesquelles vous pouvez choisir. Vous pouvez également utiliser le menu contextuel d'un artefact Statemachine Exécutable lui-même pour lancer la simulation.

Contrôler la vitesse

Le contrôle Simulation fournit un réglage de vitesse. Vous pouvez l'utiliser pour ajuster la vitesse à laquelle la simulation s'exécute. La vitesse est représentée par une valeur comprise entre 0 et 100 (une valeur plus élevée est plus rapide). Une valeur de zéro entraînera l'arrêt de la simulation après chaque étape ; cela nécessite l'utilisation des commandes de la barre d'outils pour parcourir manuellement la simulation.

Notation pour States Actif

Pendant l'exécution du Statemachine Exécutable , les diagrammes Statemachine pertinents sont affichés. L'affichage est mis à jour à la fin de chaque cycle d'étape jusqu'à l'achèvement. Vous remarquerez que seul l' State actif de l'instance effectuant une étape est mis en surbrillance. Les autres States restent dans le flou.

Il est facile d'identifier quelle instance se trouve dans quel State , car les States portent le nom de toute instance se trouvant actuellement dans cet État particulier. Si deux ou plusieurs propriétés Artifact du même type partagent le même State , l' State aura une étiquette distincte pour chaque nom de propriété.

Générer Diagramme

Après avoir terminé la simulation d'un Statemachine Exécutable , vous pouvez générer un diagramme de synchronisation à partir de la sortie. Pour faire ça:

Dans la barre d'outils de la fenêtre Simulation , cliquez sur 'Outils | Générer un Timing Diagramme '.

Génération de code pour Statemachines Exécutables

Le code généré pour un Statemachine Exécutable est basé sur sa propriété de langage. Il peut s'agir de Java, C, C++, C# ou JavaScript . Quel que soit le langage, Enterprise Architect génère le code approprié, qui est immédiatement prêt à être construit et exécuter . Aucune intervention manuelle n'est nécessaire avant de l' exécuter . En fait, après la génération initiale, n'importe quelle Statemachine Exécutable peut être générée, construite et exécutée d'un simple clic.

Langue prise en charge

Un Statemachine Exécutable supporte la génération de code pour ces langages de plateforme :

- Microsoft Natif C/C++
- .NET (C#)
- Scriptant (JavaScript)
- Oracle Java (Java)

À partir Enterprise Architect Release 14.1, la génération de code est prise en charge sans dépendance à l'environnement de simulation (compilateurs). Par exemple, si Visual Studio n'est pas installé, vous pouvez toujours générer du code à partir du modèle et l'utiliser dans votre propre projet. Les compilateurs sont toujours nécessaires si vous souhaitez simuler des modèles dans Enterprise Architect .

Environnement Simulation (paramètres Compilateur)

Si vous souhaitez simuler le modèle Statemachine Exécutable dans Enterprise Architect , ces plateformes ou compilateurs sont requis pour les langages :

Plateforme linguistique	Exemple de chemin de framework
Microsoft natif (C/C++)	C:\Program Files (x86)\Microsoft Visual Studio 12.0 C:\Program Files (x86)\Microsoft Visual Studio\2017\ Professional (ou autres éditions)
.NET (C#)	C:\ Windows \Microsoft.NET\Framework\v3.5 (ou supérieur)
Scriptant (JavaScript)	N / A
Oracle Java (Java)	C:\Program Files (x86)\Java\jdk1.7.0_17 (ou supérieur)

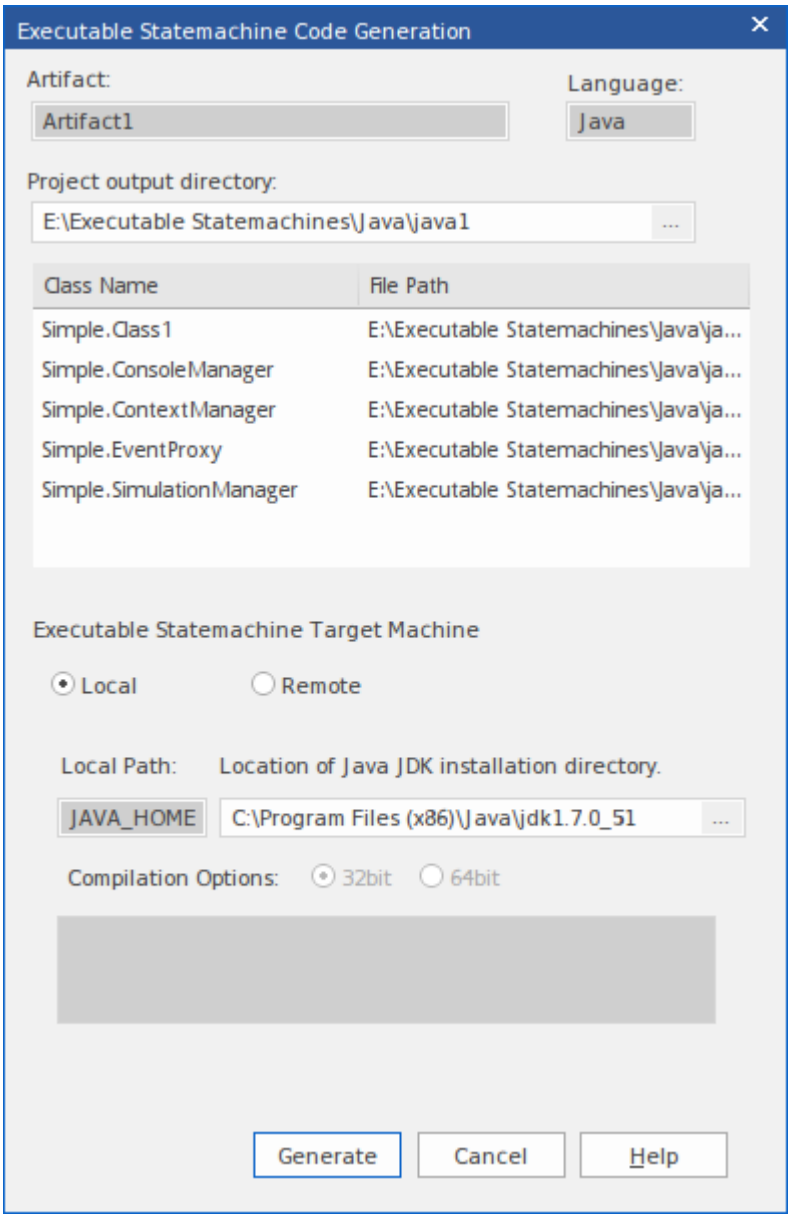
Accéder

Ruban	Simuler > États Exécutables > Statemachine > Générer , Build et Exécuter ou Simuler > États Exécutables > Statemachine > Générer

Génération de code

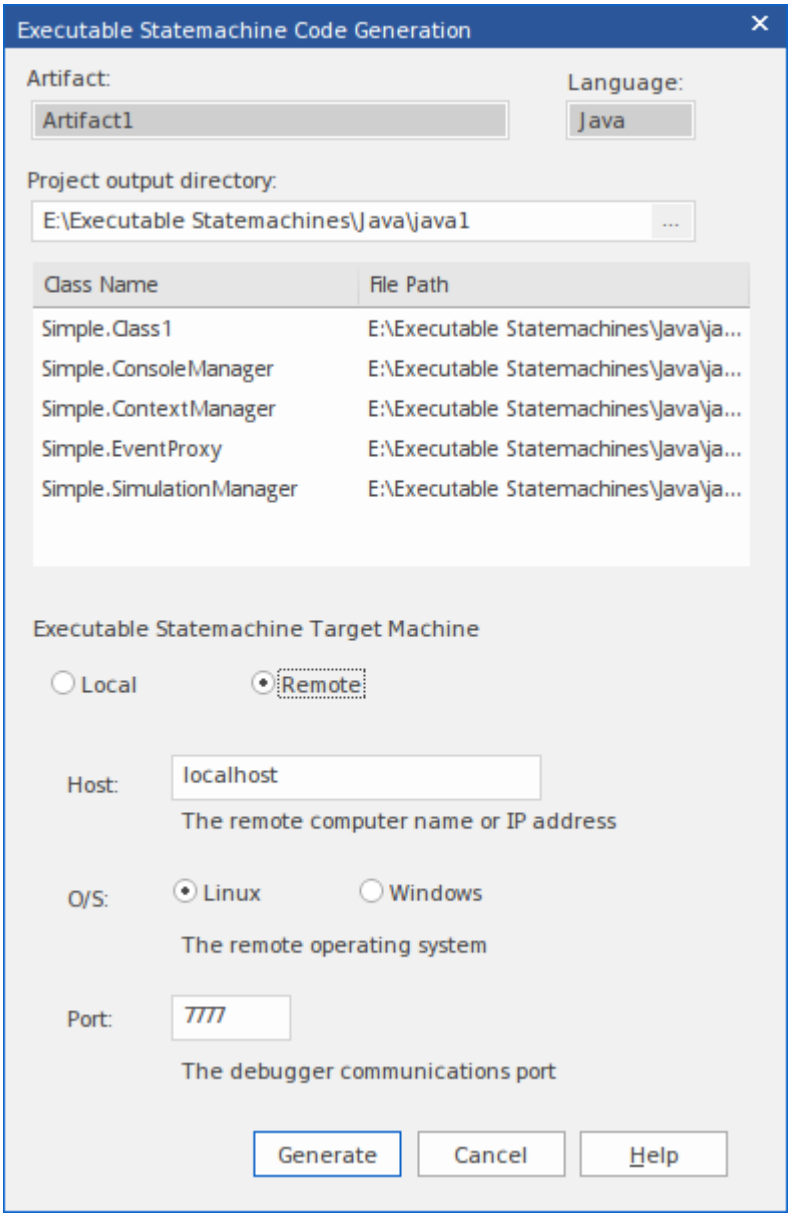
Les options du ruban « Simulate > États Exécutables > Statemachine » fournissent des commandes permettant de générer du code pour la Statemachine . Sélectionnez d'abord l'artefact Statemachine Exécutable , puis utilisez l'option du ruban pour générer le code. La dialogue « Executable Statemachine Code Generation » affichée dépend du langage de code.

Génération de code (Java sous Windows)



Répertoire de sortie du projet	Affiche le répertoire dans lequel les fichiers de code générés seront stockés. Si nécessaire, cliquez sur le bouton à droite du champ pour rechercher et sélectionner un autre répertoire. Les noms des classes générées et leurs chemins de fichiers source sont affichés après cela.
Machine cible Statemachine exécutable	Sélectionnez l'option « Local ».
JavaJDK	Entrez le répertoire d'installation du Java JDK à utiliser.

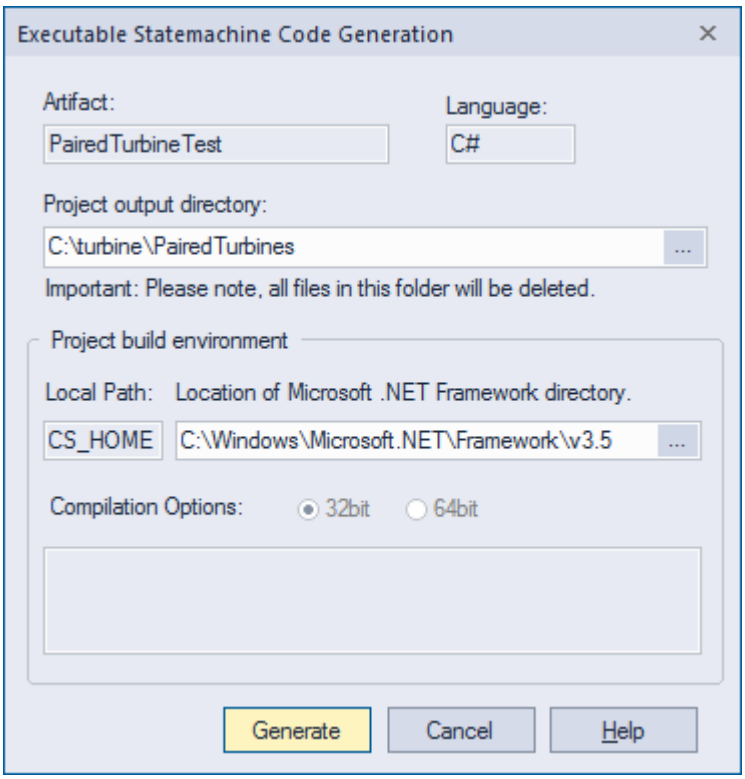
Génération de code (Java sur Linux)



Répertoire de sortie du projet :	Affiche le répertoire dans lequel les fichiers de code générés seront stockés. Si nécessaire, cliquez sur le bouton à droite du champ pour rechercher et sélectionner un autre répertoire. Les noms des classes générées et leurs chemins de fichiers source sont affichés lorsque le chemin est modifié.
Machine cible Statemachine exécutable	Sélectionnez l'option « À distance ».
Système opérateur	Sélectionnez Linux.
Port	Il s'agit du port du débogueur à utiliser. Vous trouverez les références à ce numéro

	de Port dans les sections ' Déboguer ' et ' DebugRun ' du Script Analyser généré.

Génération de code (autres langues)



En même temps, la fenêtre System Output s'ouvre sur la page ' Statemachine Exécutable Output', sur laquelle les messages de progression, les avertissements ou les erreurs sont affichés lors de la génération du code.

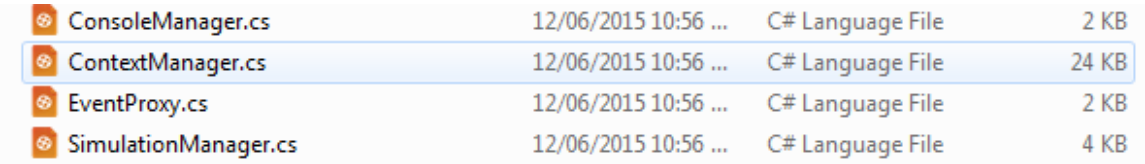
Dans la dialogue ' Statemachine Exécutable Code Generation', les champs 'Artifact' et 'Language' affichent le nom de l'élément et la langue de codage tels que définis dans la dialogue ' Propriétés ' de l'élément.

Champ/Option	Description
Répertoire de sortie du projet	Affiche le répertoire dans lequel les fichiers de code générés seront stockés. Si nécessaire, cliquez sur le bouton à droite du champ pour rechercher et sélectionner un autre répertoire.
Environnement de construction de projet	Les champs et informations de ce panneau varient en fonction de la langue définie dans l'élément Artifact et dans le script. Cependant, chaque langage pris en charge offre une option pour définir le chemin vers les frameworks cibles requis pour construire et exécuter le code généré ; des exemples sont présentés dans la section *Langues prises en charge* de cette rubrique. Ce chemin, et son ID de chemins locaux, sont définis dans la dialogue 'Chemins locaux' et affichés ici dans la dialogue ' Statemachine Exécutable Code Generation'.

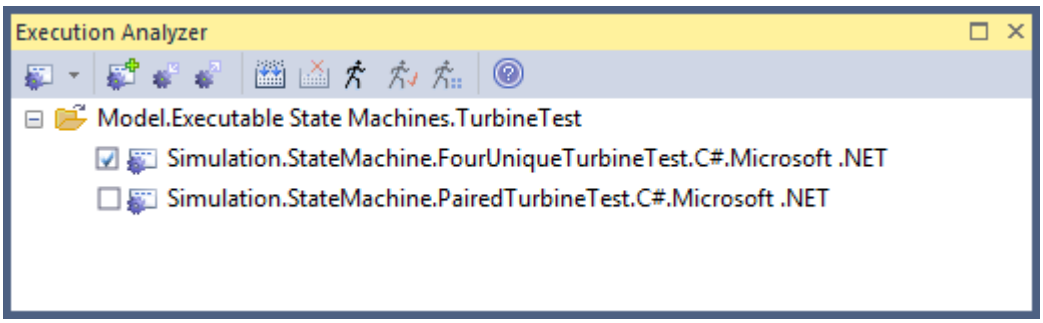
Générer

Cliquez sur ce bouton pour générer le code Statemachine . La génération de code écrasera tous les fichiers existants dans

le répertoire de sortie du projet. L'ensemble de fichiers comprendra tous les fichiers requis, y compris ceux de chaque classe référencée par le Statemachine .



Chaque Statemachine Exécutable généré générera également un script Analyseur d'Exécution , qui est le script de configuration pour construire, exécuter et déboguer le Statemachine Exécutable .



Code de construction

Le code généré par un Statemachine Exécutable peut être construit par Enterprise Architect de trois manières.

Méthode	Description
Ribbon Générer , Build et Exécuter Commande	Pour le Statemachine Exécutable sélectionné, génère à nouveau l'intégralité de la base de code. Le code source est ensuite compilé et la simulation lancée.
Commande de création de ruban	Compile le code qui a été généré. Cela peut être utilisé directement après la génération du code, si vous avez apporté des modifications à la procédure de construction (le script Analyzer) ou modifié le code généré d'une manière ou d'une autre.
Analyseur d'Exécution Script	Le script Analyseur d'Exécution généré comprend une commande pour construire le code source. Cela signifie que lorsqu'il est actif, vous pouvez créer directement à l'aide du raccourci intégré Ctrl+Shift+F12.
Créer une sortie	Lors de la construction, toutes les sorties sont affichées sur la page « Construire » de la fenêtre Sortie système. Vous pouvez double-cliquer sur n'importe quelle erreur du compilateur pour ouvrir un éditeur de source sur la ligne appropriée.

Tirer parti du code existant

Statemachines Exécutables générés et exécutés par Enterprise Architect peuvent exploiter le code existant pour lequel aucun modèle de classe n'existe. Pour ce faire, vous devez créer un élément Class abstrait nommant uniquement les opérations à appeler dans la base de code externe. Vous créeriez ensuite une généralisation entre cette interface et la classe Statemachine , en ajoutant manuellement les liens requis dans le script Analyzer. Pour Java, vous pouvez ajouter des fichiers .jar au chemin de classe. Pour le code natif, vous pouvez ajouter un .dll au lien.

Modélisation Statemachines Exécutables

La plupart du travail requis pour modéliser une Statemachine Exécutable est modélisation standard basée sur UML des modèles de classes et State , bien qu'il existe quelques conventions qui doivent être respectées pour garantir une base de code bien formée. La seule nouvelle construction est l'utilisation d'un élément Artifact stéréotypé pour former la configuration d'une instance ou d'un scénario Statemachine Exécutable . L'artefact est utilisé pour spécifier des détails tels que :

- Le langage de code (JavaScript , C# , Java, C++ dont C)
- Les classes et Statemachines impliquées dans le scénario
- Les spécifications de l'instance, y compris l'état d'exécution ; note que cela pourrait inclure plusieurs instances de la même Statemachine , par exemple lorsqu'une classe « Joueur » est utilisée deux fois dans une simulation de match de tennis.

Outils et objets Modélisation de base pour Statemachines Exécutables

Ce sont les principaux éléments modélisation utilisés lors de la construction Statemachines Exécutables .

Type d'élément	Description
Classes et Diagrammes de classes	Les classes définissent les types object pertinents pour la ou les Statemachine modélisées. Par exemple, dans un scénario simple de match de tennis, vous pouvez définir une classe pour chacun d'un joueur, d'un match, d'un Hit et d'un arbitre. Chacun aura sa ou ses propres Statemachine et, au moment de l'exécution, sera représenté par des instances object pour chaque entité impliquée. Voir le *Guide Modélisation UML* pour plus d'informations sur les classes et diagrammes de classes.
Statemachines	Pour chaque classe que vous définissez et qui aura un comportement dynamique dans un scénario, vous définirez généralement une ou plusieurs Statemachines UML . Chaque Statemachine déterminera le comportement juridique basé sur l'État approprié pour un aspect de la classe propriétaire. Par exemple, il est possible d'avoir une Statemachine qui représente l'état émotionnel d'un joueur, une qui suit sa forme physique et ses niveaux d'énergie actuels, et une autre qui représente son état de victoire ou de défaite. Toutes ces Statemachines seront initialisées et démarrées lorsque le scénario Statemachine commencera à s'exécuter.
Artefact Statemachine Exécutable	Cet artefact stéréotypé est l'élément central utilisé pour spécifier les participants, la configuration et les conditions de démarrage d'un Statemachine Exécutable . Du point de vue du scénario, il est utilisé pour déterminer quelles Instances (de Classes) sont impliquées, quels événements elles peuvent Déclencheur et s'envoyer les unes aux autres, et dans quelles conditions de départ elles opèrent. Du point de vue de la configuration, l'artefact est utilisé pour établir le lien vers un script d'analyseur qui déterminera le répertoire de sortie, le langage de code, le script de compilation et similaire. Un clic droit sur l'Artefact vous permettra de générer, construire, compiler et visualiser l'exécution en temps réel de vos Statemachines .

Constructions Statemachine prises en charge

Ce tableau détaille les constructions Statemachine prises en charge et toutes les limitations ou contraintes générales pertinentes pour chaque type.

Construction	Description
Statemachines	• Simple Statemachine : la Statemachine a une région • Statemachine orthogonale : la Statemachine contient plusieurs régions Sémantique d'activation de la région de niveau supérieur (propriété de Statemachine) : **Activation par défaut :** lorsque la Statemachine commence à s'exécuter. **Entrée du point d'entrée :** transitions du point d'entrée vers les sommets des régions contenues. • *Note 1 : Dans chaque région de la Statemachine possédant le point d'entrée, il existe au plus une seule transition du point d'entrée vers un sommet au sein de cette région.* • *Note 2 : Cette Statemachine peut être référencée par un State de sous-machine - les références de points de connexion doivent être définies dans l' State de sous-machine en tant que sources/cibles de transitions ; la référence du point de connexion représente une utilisation d'un point d'entrée/sortie défini dans la Statemachine et référencé par l' State de la sous-machine* **Statemachines multiples :** L'ordre de listage dans la fenêtre Navigateur détermine l'ordre d'exécution. • Lorsqu'un State de sous-machine est impliqué, il peut y avoir plusieurs Statemachines sous la classe. • Utilisez les flèches Monter ou Descendre dans la barre d'outils de la fenêtre Navigateur pour ajuster l'ordre des Statemachines ; celui du haut sera défini comme Statemachine principal **Non supporté** • Statemachine de protocole • Redéfinition Statemachine
States	• State simple : n'a pas de sommets ni de transitions internes • State composite : contient exactement une région • State orthogonal : contient plusieurs régions • Submachine State : fait référence à une Statemachine entière
Entrée State composite	• Entrée par défaut • Entrée explicite • Entrée d'historique peu profonde • Entrée dans l'historique approfondi • Entrée du point d'entrée
Sous-états	• Sous-états et sous-états imbriqués La sémantique d'entrée et de sortie, où la transition comprend plusieurs niveaux d'états imbriqués, obéira à l'exécution correcte des comportements imbriqués (tels que OnEntry et OnExit).
Support des transitions	• Transition externe • Transition locale • Transition interne (dessinez une transition personnelle et changez le type de transition en interne) • Achèvement Transition et Achèvement Événements • Gardes de transition

	• Transitions composées • Priorités de tir et algorithme de sélection Pour plus de détails, reportez-vous à l' *OMG UML Spécification* .
Déclencheur et Événements	Un Statemachine Exécutable supporte la gestion des événements pour les signaux uniquement. Pour utiliser les types d'événements d'appel, de synchronisation ou de changement, vous devez définir un mécanisme externe pour générer des signaux basés sur ces événements.

Signal	Attributes peuvent être définis dans Signaux ; la valeur des attributs peut être utilisée comme arguments d'événement dans Transition Gardes et Effets .
	Par exemple, voici le code défini dans l'effet d'une transition en C++ :
	si (signal-> signalEnum == ENUM_SIGNAL2)
	{
	int xVal = ((Signal2*)signal)->maVal;
	}
	Signal2 est généré sous la forme de ce code :
	classe Signal2 : signal public{
	publique:
	Signal2(){};
	Signal2(std::vector<String>& lstArguments);
	int maVal;
	} ;
	Note : De plus amples détails peuvent être trouvés en générant un Statemachine Exécutable et en se référant au fichier 'EventProxy' généré.

Initial	Un pseudo-état initial représente un point de départ pour une région. C'est la source d'au plus une transition ; il peut y avoir au plus un sommet initial dans une région.
Régions	**Activation par défaut et activation explicite :** Les transitions se terminent sur l' State contenant : • Si un Pseudoétat initial est défini dans la Région : **Activation par défaut** • Si aucun pseudo-état initial n'est défini, la région restera inactive et l' State contenant est traité comme un State simple. • Si la transition se termine sur l'un des sommets contenus dans la région : **activation explicite** , entraînant l'activation par défaut de toutes ses régions orthogonales, à moins que ces régions ne soient également saisies explicitement (plusieurs régions orthogonales peuvent être saisies explicitement en parallèle via des transitions provenant de la même région). Pseudo-état de fourche) Par exemple, si trois régions sont définies pour un State orthogonal et que *RégionA* et *RégionB* ont un pseudo-état initial, alors *RégionC* est explicitement activée. L'activation par défaut s'applique à *RegionA* et *RegionB* ; l' State contenant aura trois régions actives.
Choix	Les contraintes de garde sur toutes les transitions sortantes sont évaluées dynamiquement, lorsque le parcours de transition composé atteint ce pseudo-état.
Jonction	Branche conditionnelle statique : les contraintes de garde sont évaluées avant

	l'exécution de toute transition composée.
Fourcher / Rejoindre	Sans thread, chaque région active se déplace d'une étape en alternance, sur la base d'un mécanisme de pool d'événements d'achèvement.
Nœuds EntryPoint/ExitPoint	Non threadé pour State orthogonal ou Statemachine orthogonal ; chaque région active avance d'un pas en alternance, sur la base d'un mécanisme de pool d'événements d'achèvement.
Nœuds d'historique	• DeepHistory : représente la configuration State active la plus récente de son State propriétaire • ShallowHistory : représente le sous-état actif le plus récent de son State contenant, mais pas les sous-états de ce sous-état.
Événements différés	Dessinez une transition automatique et changez le *type* de transition en Interne. Type 'différer();' dans le champ « Effet » pour la transition.
Références des points de connexion	Une référence de point de connexion représente une utilisation (dans le cadre d'un Submachine State) d'un point d'entrée/sortie défini dans la Statemachine référencée par le Submachine State . Les références de point de connexion d'un State de sous-machine peuvent être utilisées comme sources et cibles de transitions. Ils représentent les entrées ou sorties de la Statemachine référencée par le Submachine State .
Comportements State	Les comportements d'entrée, de doActivity et de sortie d' State sont définis comme des opérations sur un State . Par défaut, vous saisissez le code qui sera utilisé pour chaque comportement dans le champ Panneau 'Code' de la fenêtre Propriétés pour l'opération Comportement. Note que vous pouvez modifier cela pour saisir le code dans le panneau 'Comportement', en personnalisant le gabarit de génération. Le comportement 'doActivity' généré sera exécuter jusqu'à son terme avant de continuer. Le code n'est pas concurrent avec d'autres comportements d'entrée ; le comportement « doActivity » est implémenté en tant que comportement « exécuter en séquence après l'entrée ».

Références à des comportements dans d'autres contextes/classes

Si l' State de la sous-machine fait référence à des éléments de comportement en dehors du contexte ou de la classe actuelle, vous devez ajouter un connecteur « import » de la classe de contexte actuelle à la classe de contexte du conteneur. Par exemple:

State de la sous-machine S1 en classe 1 fait référence à Statemachine ST2 en classe 2.

Par conséquent, nous ajoutons un connecteur « import » de Class1 vers Class2 afin que la génération de code Statemachine Exécutable génère correctement le code pour Submachine State S1. (Sur la classe 1, cliquez sur la flèche Quick Linker et faites glisser vers la classe 2, puis sélectionnez « Importer » dans le menu des types de connecteurs.)

Réutilisation d'artefacts Statemachine Exécutable

Vous pouvez créer plusieurs modèles ou versions d'un composant à l'aide d'un seul artefact exécutable. Un artefact représentant une résistance, par exemple, pourrait être réutilisé pour créer à la fois une résistance en feuille et une résistance bobinée. Cela est probablement le cas pour des objets similaires qui, bien que représentés par le même classificateur, présentent généralement des états exécuter différents. Une propriété nommée 'resistorType' prenant la

valeur 'wire' plutôt que 'foil' pourrait suffire d'un point de vue modélisation . Les mêmes Statemachines peuvent ensuite être réutilisées pour tester les changements de comportement qui pourraient résulter d'une variation de l'état d'exécution. Voici la procédure :

Étape	Action
Créer ou ouvrir diagramme de composants	Ouvrez un diagramme de composants sur lequel travailler. Il peut s'agir du diagramme contenant votre artefact original.
Sélectionnez le Statemachine Exécutable à copier	Recherchez maintenant l'artefact Statemachine Exécutable original dans la fenêtre Navigateur .
Créer le nouveau composant	Tout en maintenant la touche Ctrl enfoncée, faites glisser l'artefact original sur votre diagramme . Deux questions vous seront posées. La réponse à la première est **Object** et à la seconde **Tout.** Renommez l'artefact pour le différencier de l'original, puis modifiez les valeurs de ses propriétés.